VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Multi-Agentní systémy a jejich propojení s webovými službami

Multi-Agent Systems and its cooperation with web services

2010 Martin Damovský

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 21. dubna 2010 .....

Mé díky patří vedoucímu této diplomové práce Ing. Michalu Radeckému za spoustu zajímavých nápadů a podnětů, kterými obohatil tuto práci. Také bych rád poděkoval Bc. Pavlovi Lobodinskému za připomínky k samotnému textu práce.

Abstrakt

Tato práce se zabývá problematikou Multi-Agentních systémů (MAS), moderním přístupem k distribuované umělé inteligenci. Multi-Agentní systémy získávají na oblibě díky svým vlastnostem, mezi které patří např. odolnost vůči výpadku klíčového prvku (žádný centrální prvek neexistuje). Také díky této zvyšující se oblibě se objevují potřeby propojit Multi-Agentní systémy s jinými technologiemi, např. webovými službami. Hlavním tématem této práce je tedy problém propojitelnosti Multi-agentních systémů a webových služeb. V první části práce je detailně vysvětlena teorie týkající se MAS, implementačního frameworku Jade (.NET verze) a webových služeb. Ve druhé části je popsán návrh a implementace modulu umožňujícího komunikaci mezi MAS a webovými službami. V závěrečné části je zmíněno několik problémů, které více či méně ovlivní další vývoj hotového modulu brány.

Klíčová slova: Multi-agentní systém, MAS, ontologie, Jade, LEAP, webové služby, SOAP, WSDL, WS-Inspection, .NET, Java

Abstract

This thesis deals with Mult-Agent system issues (MAS), modern approach to distributed artifical intelligence. Multi-agent systems obtains popularity thanks to its properties, for example resistance for central item's outage (there is no central item). Thanks to this increasing popularity, there is a need to interconnect Multi-Agent systems with other technologies like a web services. Therefore the main topic of this thesis is problem of connectivity between Multi-Agent systems and web services. In the first part of thesis is explained theory regarding MAS, framework Jade (.NET version) and web services. In the second part is described design and implementation of modul providing communication between MAS and web services. In the final part is mentioned a few problems that can more or less affects future implementation of completed gateway modul.

Keywords: Multi-agent system, MAS, ontology, Jade, LEAP, webservices, SOAP, WSDL, WS-Inspection, .NET, Java

Seznam použitých zkratek a symbolů

ACL	–	Agent Communication Language
AID	–	Agent Identifier File
AMS	–	Agent Management System
B2B	–	Business to Business
DF	–	Directory Facilitator
FIPA	–	Foundation for Intelligent Physical Agent
GUI	–	Graphical User Interface
HTTP	–	Hypertext Transfer Protocol
HTTPS	–	Hypertext Transfer Protocol Secure
IIOP	–	Internet Inter-ORB Protocol
JADE	–	Java Agent Development Framework
KQML	–	Knowledge Query Manipulation Language
LEAP	–	Lightweight Extensible Agent Platform
MAS	–	Multi agent system
MIDP	–	Mobile Information Device Profile
O2A	–	Object to Agent
SOAP	–	Simple Object Access Protocol
SSL	–	Secure Socket Layer
SVN	–	Subversion
UDDI	–	Universal Description, Discovery and Integration
URL	–	Uniform Resource Locator
W3C	–	World Wide Web Consortium
WSDL	–	Web Service Description Language
WSIG	–	Web Services Integration Gateway
WSIL	–	Web Services Inspection Language (WS-Inspection)
XML	–	Extensible Markup Language

Obsah

Seznam obrázků

Seznam výpisů zdrojového kódu

1 Úvod

Multi-Agentní systémy tvoří velice zajímavé odvětví distribuované umělé inteligence, které si získává čím dál tím větší popularitu. Každý Multi-agentní systém tvoří vlastní *svět*, kde mohou agenti žít, komunikovat či dokonce umírat dle předem stanovených pravidel. Nicméně tento *svět* je poněkud uzavřený. Komunikace s vnějškem je obvykle omezená na možnosti implementačního rámce (frameworku). Tato vlastnost se také stala podnětem pro vznik této diplomové práce, která si klade za cíl **poskytnout komunikační rozhraní mezi MAS a ostatními aplikacemi**. Výstup této práce má potenciál stát se součástí rozsáhlejších projektů v rámci Laboratoře inteligentních systémů (LabIS, `http://labis.vsb.cz`).

Během prvotního studia MAS a frameworku Jade bylo zjištěno, že problém propojitelnosti MAS a webových služeb je celkem nový - až na jednu výjimku neexistovalo žádné řešení, které by umožnilo propojit MAS (Jade) a webové služby. Zmíněnou výjimkou byl projekt WSIG[1], který v té době (jaro 2008) umožňoval pouze komunikaci s agentem (reprezentovaným webovou službou). Tento nedostatek přispěl ke snaze zveřejnit výsledek této práce jako rozšíření frameworku Jade.

Během práce jsem vycházel z mnoha publikací týkající se Multi-agentních systémů a jejich implementací ve frameworku Jade, které jsou uvedené v sekci **Literatura**. Zdrojem mnoha užitečných informací byly také zdrojové kódy frameworku Jade, které jsou volně dostupné v SVN repository[2].

Čtenáře může v textu překvapit použití angličtiny pro popis obrázků či tabulek. Část této publikace bude zveřejněna rovněž v anglickém jazyce, a tudíž jsem se rozhodl používat ve všech grafických materiálech právě angličtinu.

1.1 Struktura práce

V prvních dvou kapitolách se zabývám technologiemi, které jsou hlavní náplní této práce - Multi agentní systémy a webové služby. Poté následuje krátké představení problematiky, tj. propojení multi-agentních systémů a webových služeb. Další kapitoly již navazují na vodopádový model vývoje aplikací - specifikace požadavků, analýza, návrh, implementace a testování. V předposlední kapitole jsou popsány možnosti použití celého řešení v praxi, stejně tak jako budoucnost celého řešení.

V sekci **Přílohy** může čtenář nalézt mnoho užitečných informací, např. návod „jak zprovoznit celé řešení“, „jak zkompilovat Jade-LEAP do DLL knihovny“ apod.

[1] `http://jade.tilab.com/dl.php?file=wsigAddOn-2.0.zip` (jaro 2008)

[2] SVN repository je dostupná na URL `https://avalon.cselt.it/svn/jade/trunk` (login = jade, password = jade)(listopad 2009)

1.2 Použité značení

- v textu se často objevují názvy tříd a metod, které jsou pro přehlednost zvýrazněné *kurzívou*,
- v některých diagramech jsou použity názvy metod a tříd. Aby bylo možné rozlišit metody a třídy frameworku Jade od ostatních (např. .NET), jsou všechny třídy a metody frameworku Jade ve všech diagramech označeny prefixem „*Jade::*“ (např. *Jade::putO2AObject()*).

1.3 Pojmenování

Každý projekt by měl mít svůj název. Výsledek této práce není výjimkou, a proto jsem zvolil název „**Jade2WS for .NET**“.

2 Úvod do problematiky

I když si to často neuvědomujeme, mnoho systémů kolem nás se vyznačuje vysokou komplexností, přičemž každý z těchto systémů se řídí svými pravidly. Hezkým příkladem je provoz na silnici, který se řídí silničními pravidly. Tato pravidla jsou závazná pro každého účastníka silničního provozu (at' už se jedná o nákladní automobil nebo jen cyklistu). Podíváme-li se na účastníky silničního provozu, můžeme o nich říct, že každý účastník má svůj vlastní cíl, kterého chce na silnici dosáhnout - taxikář veze turistu na letiště, slečna jede autem do práce, policie spěchá k automobilové nehodě, apod.

Chtěli bychom-li takovýto komplexní systém simulovat v počítači, dojdeme nejspíš k názoru, že potřebujeme nějak vytvořit navzájem nezávislé prvky schopné samostatně jednat a konat s cílem dosáhnout určitého cíle. Navíc by byla aktivita těchto prvků striktně omezena pravidly a prostředím (např. pravidly silničního provozu). A v neposlední řadě nesmíme zapomenout na nutnost či možnost vzájemné spolupráce (prakticky každý automobilový řidič již někdy využil spolupráce s ostatními řidiči, např.v případě uvíznutí v závěji, apod.).

Zmíněné „nezávislé prvky" jsou ve světě IT často nazývány jako „agenti". Tito agenti tvoří společně s prostředím a pravidly jednotný celek jménem „Multi-agentní systém". Multi-agentní systém je obecný výpočetní model sloužící k simulaci či řešení právě tak složitých systémů jako např. silniční provoz.

3 Multi-agentní systémy

3.1 Co je to Agent

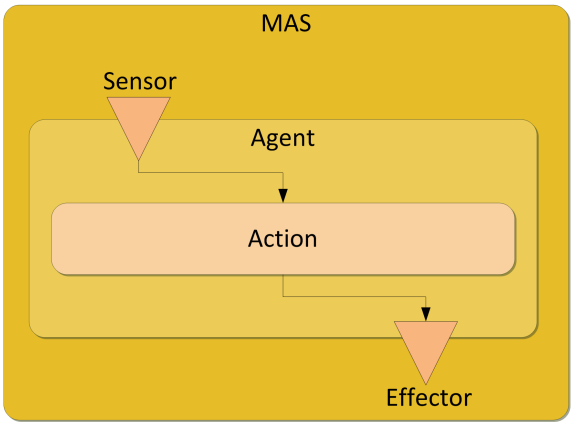
Definice 3.1 *„Agent je entita zkonstruována za účelem kontinuálně a do jisté míry autonomně plnit své cíle v adekvátním prostředí na základě vnímání prostřednictvím senzorů a prováděním akcí prostřednictvím efektorů. Agent přitom ovlivňuje podmínky v prostředí tak, aby se přibližoval k plnění cílů“[1].*

Agent je tedy jen jakýsi zástupný prvek (vzpomeňme na účastníky silničního provozu, viz kapitola 2). Nikoho proto nepřekvapí následující stručná charakteristika agenta:

- je autonomní - tzn. jedná bez závislosti na okolí (je nezávislý),
- je reaktivní - tzn. je schopen reagovat na změny prostředí,
- je proaktivní - tzn. je schopen ovlivňovat okolní prostředí za účelem dosažení svých cílů,
- má sociální schopnosti - tzn. dokáže spolupracovat s ostatními agenty za účelem dosažení svého cíle,

Každý agent je složen z několika základních částí:

- senzor - prostřednictvím senzoru agent vnímá své okolní prostředí (např.přijímá komunikaci od okolních agentů),
- akce (často označované jako „chování/behaviour“) - jasně definovaná posloupnost kroků, která je obvykle zinicializována senzorem,
- efektor - prostřednictvím efektoru agent jedná s okolím (např.komunikuje s okolními agenty).



Obrázek 1: Grafické znázornění obecné architektury agenta

Některé z typických vlastností agentů, jako např. způsob komunikace mezi agenty, často připomínají objekty z Objektově-orientovaného programování. Podíváme-li se ale do hloubky, zjistíme, že existují podstatné rozdíly[1]:

1. Komunikace
   - v případě komunikace mezi objektem A a objektem B je nutné, aby objekt A zavolal metodu objektu B, tzn. komunikuje se voláním metod,
   - v případě komunikace mezi agenty se komunikuje předáváním zpráv ve vyšších komunikačních jazycích jako např. ACL, navíc je tato komunikace asynchronní.
2. Vlákna
   - více objektů obvykle sdílí jedno vlákno,
   - každý agent je reprezentován svým vlastním vláknem (je více autonomní).
3. Zapouzdřenost
   - objekt zapouzdřuje metody a proměnné,
   - agent zapouzdřuje tzv.chování, kterých může být pro jednoho agenta několik.

3.2 Co je to Multi-agentní systém

Definice 3.2 *„Multi-agentní systém (MAS) je výpočetní systém, ve kterém agenti spolupracují tak, aby bylo dosaženo nějakého kolektivního či individuálního cíle.“*[3].

Oblast Multi-agentních systémů (dále jen MAS) lze spojit s vývojem Distribuované umělé inteligence. První náznaky využití distribuovaného výpočetního prostředí se objevily již v 80.letech 20.století. Kořeny MAS lze najít ve snaze rozdělit navrhovaný systém na samostatné autonomní jednotky schopné vzájemné interakce. Jednou z motivací pro dekompozici velkých systémů na autonomní jednotky je složitost a komplexnost některých aplikací jako např. systémy řízení letového provozu, řízení bezpilotních letadel či robotický fotbal.

Charakteristickou vlastností MAS je decentralizace řízení, tzn. neexistuje jeden centrální prvek, který řídí ostatní. V MAS má každý agent svou roli a je schopen vyjednávat s okolními agenty a případně se domlouvat na další spolupráci.

Pokud uvolníme uzdy své fantazii, můžeme si multi-agentní systém představit jako mraveniště, které poskytuje prostor pro život a spolupráci miliónům mravenců. Každý mravenec se umí rozhodovat sám za sebe, ale pokud by se vydal sám mimo mraveniště, pravděpodobně by umřel hlady. Síla těchto miniaturních tvorů je v jejich vzájemné spolupráci, jedině tak mohou docílit velkých výsledků. Obdobné je to i s agenty v multi-agentním systému.

[3] Zdroj: `http://agtivity.com/def/multi_agent_system.htm`(duben 2010)

3.3 Multi-agentní systémy v praxi

Podíváme-li se na již existující implementaci MAS, zjistíme, že např. pro systém řízení letového provozu vyvíjí Agent Technology Center ČVUT Praha systém s názvem AgentFly. „AgentFly je softwarový prototyp pro řízení letového provozu, který podporuje koncept volného letu (free flight). Každé letadlo v systému je modelováno jako nezávislá entita, která muže hostit několik inteligentních agentů. Každá entita je zodpovědná za naplánování a provedení letového plánu. Během letu detekují agenti případné kolize a zahájí vzájemné peer-to-peer vyjednávání, při kterém inteligentním způsobem pře-plánují aktuální letové plány na bezkolizní ".[4]

Možnosti využití MAS jsou obrovské a atraktivita této problematiky dává naději, že se v horizontu deseti let setkáme s masivním využitím systémů pro:

- řízení dopravy ve velkých městech,
- spolehlivější předpovídání přírodních katastrof,
- řízení a plánování výroby ve velkých výrobních společnostech,
- apod.

3.4 Framework JADE

JADE (Java Agent Development Framework) je framework plně napsaný v Javě a slouží k implementaci Multi-Agentních systémů dle specifikace FIPA [3]. Distributorem tohoto frameworku je Telecom Italia, který jej distribuuje jako open source pod licencí LGPL (Lesser General Public License Version 2).

První řádky kódu tohoto frameworku se začaly psát v roce 1998. Nicméně framework Jade se ve svých prvních verzích neubíral směrem plnohodnotné platformy pro vývoj MAS. Jako podnět pro jeho vývoj se v dostupných pramenech uvádí „potřeba validace specifikací FIPA"[8]. Obrat přišel až po finanční podpoře ze strany Evropské komise Cordis (projekt AC317 FACTS [5]).

Nyní, v roce 2010, má framework Jade rozsáhlou komunitu vývojářů, vědeckých pracovníků a obchodních společností. Tato komunita žene vývoj frameworku kupředu mílovými kroky o čemž svědčí i množství rozšíření (add-ons)[6].

V době psaní této práce je na oficiálních stránkách http://jade.tilab.com dostupná verze 3.7.

[4] Zdroj: `http://fel.cvut.cz/cz/vv/tymy/atg.html` (leden 2010)

[5] Podrobnosti týkající se projektu AC317 jsou dostupné na URL `http://cordis.europa.eu/infowin/acts/rus/projects/ac317.htm` (leden 2010)

[6] Add-ons jsou dostupné na URL `http://jade.tilab.com/community-3rdpartysw.htm` (leden 2010)

3.4.1 Vlastnosti frameworku JADE

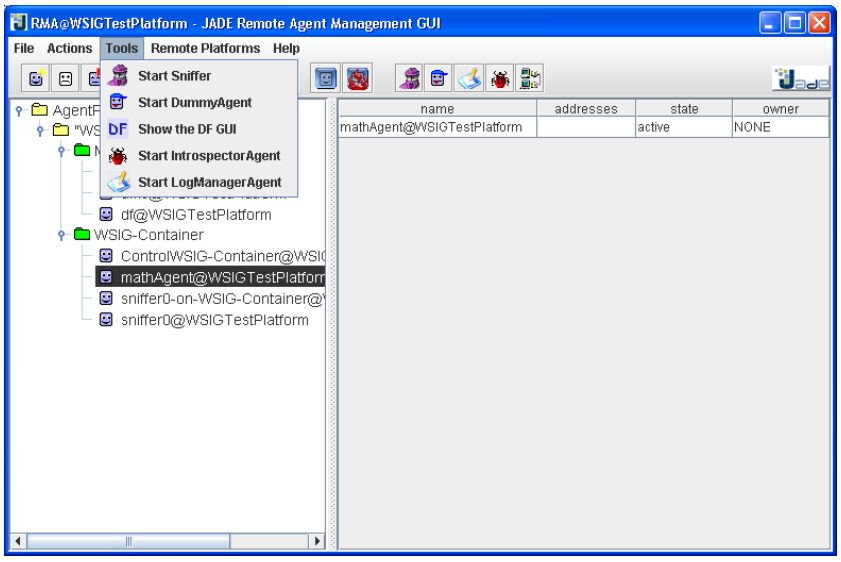
Framework Jade poskytuje vývojářům nástroje pro vývoj MAS, které zahrnují:

- runtime prostředí, ve kterém mohou být agenti spuštěni,
- knihovny tříd pro vývoj vlastních agentů,
- grafické rozhraní pro administraci a monitoring MAS, viz obrázek 2.

Instance runtime prostředí se nazývá kontejner, přičemž se rozlišuje mezi Hlavním kontejnerem a Agent kontejnerem. Hlavní kontejner musí být spuštěn jako první, každý další Agent kontejner se musí připojit k Hlavnímu kontejneru. Jednotlivé kontejnery mohou být umístěné na různých počítačích, případně mobilních zařízeních.

Hlavní kontejner poskytuje několik služeb, díky kterým se tak odlišuje od Agent kontejnerů:

- Agent Management Service (AMS)
  - služba zodpovědná za správu agentů jako např. vytváření, mazání, správa migrace agentů, apod.
- Directory Facilitator (DF)
  - poskytuje tzv.yellow service (často překládané jako „zlaté stránky“). Prostřednictvím DF lze dohledat informace o dostupných agentech, tzn. jedná se o jakýsi „adresář agentů“.



Obrázek 2: Grafické rozhraní pro správu a monitoring MAS v JADE

Samotný framework Jade neposkytuje žádnou podporu pro umělou inteligenci jednotlivých agentů. Nicméně jednotliví agenti mohou být naimplementování tak, aby simulovali umělou inteligenci např. s použitím genetických algoritmů, neuronových sítí, apod.

3.4.2 Ontologie

Během komunikace mezi agenty se využívá obecného principu popisu dat, který se nazývá ontologie. Ontologie, dle chápání informatiky, znamená popis jazyka na sémantické úrovni. Ontologie popisuje:

- prvky jazyka a jejich význam,
- vztahy mezi těmito prvky.

V posledních letech se často mluví o ontologii ve spojitosti se sémantickým webem, který by měl být rozšířením dnešního internetu. Cílem sémantického webu je dát informacím význam. Tim Berners-Lee, vynálezce WorldWideWeb (WWW), v roce 2001 upozornil, že stávající internet je v podstatě jen směsice stránek bez významu.

Framework Jade samozřejmě umožňuje nadefinovat si vlastní ontologii. Definice ontologie se provádí na úrovni zdrojových kódů, viz výpis zdrojového kódu 1.

Obecně lze říci, že ontologie má podobný význam jako definice metod ve WSDL.

```
1  public class MyOntology : Ontology
2  {
3      public MyOntology(): base(ONTOLOGY_NAME, BasicOntology.getInstance())
4      {
5          add(new AgentActionSchema(ACTION_NAME), java.lang.Class.forName(...));
6          add(new ConceptSchema(RESPONSE), java.lang.Class.forName(...));
7
8          AgentActionSchema request = (AgentActionSchema)getSchema(ACTION_NAME);
9          request.add(PARAM1, (PrimitiveSchema)getSchema(BasicOntology.STRING));
10         request.add(PARAM2, (PrimitiveSchema)getSchema(BasicOntology.STRING));
11         request.setResult((ConceptSchema)getSchema(RESPONSE));
12
13         ConceptSchema response = (ConceptSchema)getSchema(RESPONSE);
14         response.add(VALUE, (PrimitiveSchema)getSchema(BasicOntology.STRING));
15     }
16 }
```

Výpis 1: Ukázka definice ontologie

3.4.3 Agent

Vytvoření agenta schopného běhu v runtime prostředí (v Hlavním nebo Agent kontejneru) je velice snadné. Stačí vytvořit třídu, která bude dědit ze třídy *jade.core.Agent* a překrýt metodu *setup()*, viz výpis kódu 2. Metoda *setup()* slouží k inicializaci agenta, zaregistrování se do DF, vytvoření spojení na databázi, vytvoření GUI, apod.

```
1 using jade.core;
2
3 public class HelloWorldAgent : Agent
4 {
5     public override void setup()
6     {
7         Console.WriteLine("Hello World!");
8     }
9 }
```

Výpis 2: Ukázka vytvoření Agenta

Takto vytvořený agent je sice schopný běhu v kontejneru, nicméně nebude prakticky nic dělat - po spuštění vypíše na svůj standardní výstup (obvykle konzole) text „Hello World!“. Abychom přinutili agenta k nějaké akci, je nutné tomuto agentu přidat chování, viz následující kapitola 3.4.4. Bez chování je agent v podstatě jen „prázdnou skořápkou“.

3.4.4 Chování (Behaviour)

Chování (behaviour) v Jade reprezentuje konkrétní úkol, kterým se má agent zabývat. Vytvoření základního chování je opět velice snadné. Stačí vytvořit třídu, která dědí z *jade.core.behaviours.Behaviour* a překrýt metodu *action()*, ve které máme prostor pro vlastní implementaci daného úkolu.

```
1 using jade.core.behaviours;
2
3 public class MyBehaviour : Behaviour
4 {
5     public override void action()
6     {
7         // do something
8     }
9
10    public override bool done()
11    {
12        return false;
13    }
14 }
```

Výpis 3: Ukázka vytvoření chování

Nesmíme zapomenout přiřadit toto chování konkrétnímu agentovi pomoci metody *addBehaviour(new MyBehaviour())*, kterou je nutné zavolat z metody setup() daného agenta.

Každý agent může mít více chování. Jednotlivá chování v podstatě nikdy neběží paralelně, jejich spouštění je iniciováno vnitřním mechanismem každého agenta, který tyto chování vybírá ze svého vnitřního zásobníku a spouští jejich metodu *action()*. Tato chování zůstávají v zásobníku do doby než jejich metoda *done()* vrátí hodnotu true, tzn. chování tak dává najevo, že svůj úkol již splnilo a není již dále potřeba. Vývojový diagram spouštění jednotlivých chování lze nalézt na obrázku 3.

Framework Jade poskytuje několik typů chování, které se liší svými vlastnostmi. Mezi hlavní tři chování lze zařadit tyto:

1. *OneShotBehaviour* - toto chování je spuštěno pouze jednou, poté jeho metoda *done()* vrátí hodnotu true.
2. *CyclicBehavour* - jak název napovídá, jedná se o tzv. cyklické chování, tzn.nikdy nekončící.
3. *Behaviour* - základní typ chování, jeho charakteristika záleží na implementaci potomka.

Z dalších chování lze zmínit např. *WakerBehaviour*, které se spouští po uplynutí předem stanovené doby. Framework Jade poskytuje celkem 15 typů chování.

Jednotlivé chování lze samozřejmě patřičně modifikovat a přizpůsobit tak konkrétnímu úkolu, který má reprezentovat.

3.5 JADE-LEAP

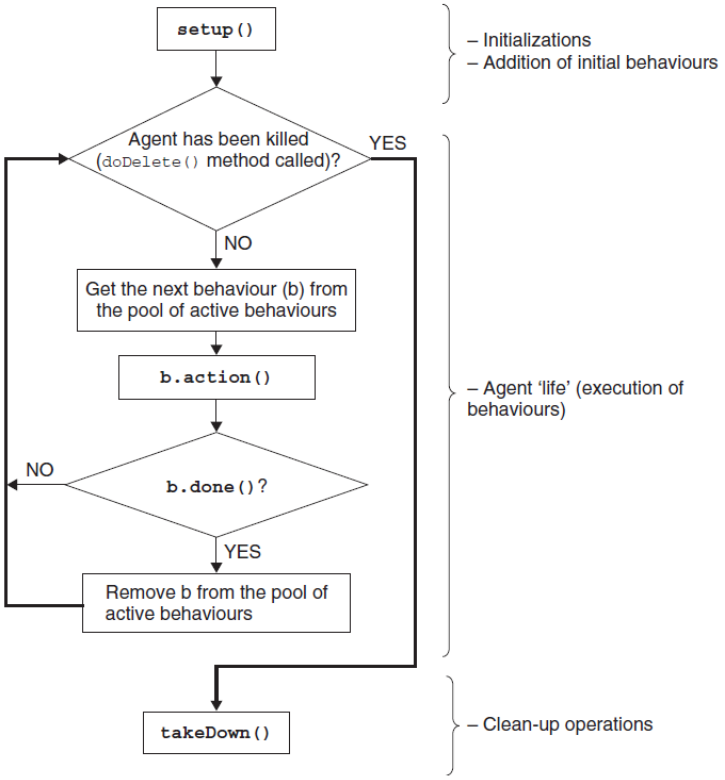
Framework Jade je dostupný pouze pro platformu Java (J2SE). Tato skutečnost významně omezuje implementaci MAS v Jade pro mobilní platformy. Proto bylo v roce 2002 vyvinuto rozšíření Jade s názvem LEAP (*Lightweight Extensible Agent Platform*). Vývoj tohoto rozšíření byl částečně financován Evropskou komisí (projekt IST1999,10211 [7]). Celkové náklady na tento projekt se vyšplhaly na 5,42 miliónů EUR. Na projektu participovaly tyto organizace: Motorola, British Telecommunications, Broadcom Eireann, Siemens AG, Telecom Italia a the University of Parma. Skupina pracující na projektu LEAP byla vedena osobou Giovanni Caire, senior project manager v Research Labs of Telecom Italia. Tento člověk je podepsán mimo jiné i pod mnoha dostupnými publikacemi pro Jade, zdrojovými kódy Jade a dokonce je i správcem a častým přispěvovatelem e-mailové diskusní skupiny Jade-News a Jade-Develop.

Poznámka 3.1 Projekt LEAP je založen na frameworku Jade, nicméně některé prameny[8] tvrdí, že LEAP částečně těží z architektury projektu ZEUS [9].

[7] Podrobnosti týkající se projektu IST1999,10211 jsou dostupné na URL `http://cordis.europa.eu/fetch?CALLER=PROJ_ICT\&ACTION=D\&DOC=28\&CAT=PROJ\&QUERY=011f9abb95a5:5dec:74cebd49\&RCN=58391` (leden 2010)

[8] zmínka o využití projektu ZEUS v LEAP je uvedena na URL `http://www.fipa.org/resources/livesystems.html` (leden 2010) a také v archívu konference na URL `http://jade.tilab.com/jade-news-archive/0013.html` (leden 2010)

[9] více informací k projektu ZEUS lze najít na URL `http://www.cs.iastate.edu/~baojie/acad/current/zeus/zeus.htm` (leden 2010)



Obrázek 3: Vývojový diagram agenta a spouštění jednotlivých chování [8]

Výsledek projektu LEAP je dostupný jako rozšíření platformy Jade a brzy začal být nazýván jako Jade-LEAP.

3.6 Hlavní rozdíly mezi Jade a Jade-LEAP

Mezi hlavní výhody projektu Jade-LEAP patří možnost implementace MAS v různých verzích Javy (J2SE, J2ME, a Personal pJava). Tyto mobilní platformy se vyznačují omezenými paměťovými zdroji a hlavně primitivní implementací síťových služeb (GPRS, apod.). Těmto vlastnostem se samozřejmě musel Jade-LEAP přizpůsobit a nahradit či úplně odstranit některé funkce a vlastnosti plnohodnotného Jade. Jeden příklad za všechny: plnohodnotný Jade používá pro komunikaci uvnitř MAS technologii RMI, která je známá svou náročností na množství používaných síťových portů. Tato technologie je zcela nevhodná pro mobilní platformy a proto Jade-LEAP využívá proprietární protokol JICP (*Jade Inter Container Protocol*).

Uživatelé projektu Jade-LEAP jsou výrazně ovlivnění i další vlastností tohoto rozšíření, a to odstraněním některých metod, které třídy frameworku Jade poskytují. Například instance třídy Ontology poskytuje metodu *getActionNames()*, které vrací seznam všech akcí definovaných v ontologii. Tato metoda je v Jade-LEAP nedostupná. Jednou

z příčin je použití prvků jazyka Java, které jsou dostupné pouze v rámci desktopové (J2SE) či serverové Javy (J2EE).

3.7 JADE-LEAP pro .NET

V polovině roku 2002 zveřejnil Microsoft programovací jazyk Visual J#, který již obsahoval podporu Javy v.1.1 pro .NET framework. Chvíli poté se v hlavě výzkumného pracovníka firmy Siemens AG Steffena Rusitschka zrodil nápad pro využití Microsoft Visual J# k portaci Jade-LEAP do prostředí platformy .NET. Výsledné řešení bylo poprvé zveřejněno v e-mailové konferenci Jade-News [10] v červenci 2002.

Celé řešení pro portaci Jade-LEAP do platformy .NET vyžaduje:

1. Microsoft Visual Studio,
2. Microsoft Visual J#,
3. speciální build script pro Ant (Java build tool),

Build script pro Ant využívá speciální task - preprocessor (naimplementovaný v javě), který projde všechny zdrojové kódy Jade a odstraní metody, které nelze převést do platformy .NET. Tyto metody jsou označeny speciálními komentáři, viz výpis zdrojového kódu 4.

```
1  //#DOTNET_EXCLUDE_BEGIN
2  public void methodName(params...) {
3      ...
4  }
5  //#DOTNET_EXCLUDE_END
```

Výpis 4: Označení metody pro nekompilování do platformy .NET

Jade-LEAP je pravidelně aktualizován dle poslední vydané verze frameworku Jade a je ke stažení na oficiálních stránkách projektu Jade.

[10] Oznámení o portaci Jade-LEAP do .NET platformy je dostupné na URL `http://jade.cselt.it/jade-news-archive/0041.html`(leden 2010)

4 Webové služby

Webové služby jsou technologie, která umožňuje zpřístupnit rozhraní aplikace komukoliv, nezávisle na platformě či fyzické vzdálenosti jednotlivých aplikací.

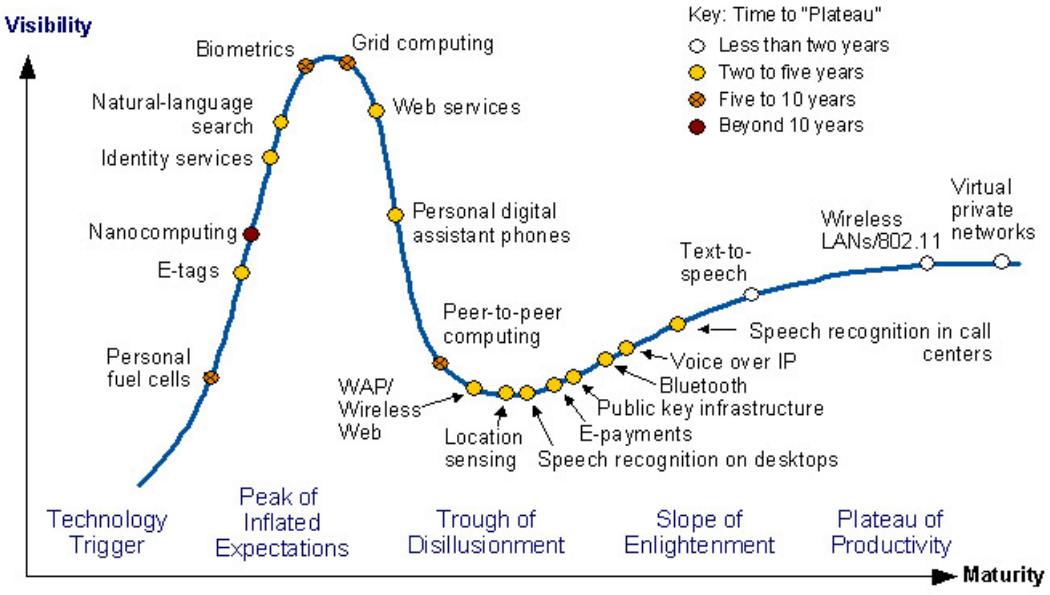
Definice 4.1 *„Webová služba je software navržený k podpoře spolupráce mezi dvěma stroji napříč počítačovou sítí. Má své rozhraní popsané strojově zpracovatelným formátem (WSDL). Ostatní systémy spolupracují s webovou službou způsobem předepsaným v tomto popisu s využitím SOAP zpráv, které jsou obvykle posílány přes HTTP protokol" [4].*

Princip webových služeb je tedy založen na dvou protokolech:

1. SOAP - pro komunikaci s webovou službou,
2. WSDL - pro popis rozhraní webové služby.

Kromě těchto technologií se často používá i registr webových služeb, např. UDDI nebo novější WS-Inspection, viz kapitola 4.3.

Koncept webových služeb je znám od dob masivního rozšiřování Internetu (cca rok 1996). V té době zveřejnil Marc Andreessen (spolumajitel firmy Netscape) svou vizi o využití komunikace mezi aplikacemi v rámci internetu pro použití v komerční sféře. Tato komunikace by měla být založena na protokolu IIOP (*Internet Inter-ORB Protocol*), který umožní požadovat službu od jiné aplikace napříč platformami.



Obrázek 4: Křivka předpovídající masivní využití webových služeb od společnosti Gartner Group z roku 2002. Zdroj: Gartner Group

V roce 2001 vydala prestižní americká analytická společnost Gartner Group studii, ve které očekává v období 2001-2005 doručení mnoha nástrojů pro vývoj webových služeb od velkých společností jako Microsoft, IBM, Sun, Software AG, Oracle a další. Tyto nástroje pro vývoj měly způsobit masivní využívání webových služeb napříč Internetem. Podle Gartner Group se v období 2002-2004 očekával obrovský nárůst využívání webových služeb zejména v komerční sféře - obzvláště v B2B sféře (viz graf na obrázku 4). Odhady této analytické společnost byly veskrze správné (alespoň co se webových služeb týče).

Dnes se webové služby staly součástí téměř každé větší aplikace a je téměř nepředstavitelné, aby aplikace neposkytovala své API skrz webové služby. Toto tvrzení dokládá i použití webových služeb v rámci Informačního systému datových schránek[11], kde jsou webové služby použity pro přístup do systému pomocí aplikací třetích stran.

4.1 Komunikace s webovou službou (SOAP)

Komunikace s webovou službou probíhá prostřednictvím XML zpráv dle protokolu SOAP (Simple Object Access Protocol). SOAP definuje tři základní elementy:

- Envelope - kořenový element,
- Header - nepovinný element sloužící například pro přenos autentizačních informací,
- Body - obsahuje elementy pro identifikaci webové služby, názvu metody a hodnoty jednotlivých parametrů volání.

V době psaní této práce je SOAP dostupný ve verzi 1.2. Jeho přesnou specifikaci lze nalézt na stránkách konsorcia W3C.

```
1 <soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
2   <soap:Body>
3     <getProductDetails xmlns="http://warehouse.example.com/ws">
4       <productID>827635</productID>
5     </getProductDetails>
6   </soap:Body>
7 </soap:Envelope>
```

Výpis 5: Ukázka zprávy definované protokolem SOAP

4.2 Popis rozhraní webové služby (WSDL)

Web Service Description Language (WSDL) je XML formát pro popis rozhraní webové služby[5]. Jedná se tedy o strojově čitelný popis metod nabízených webovou službou.

Téměř každý framework využívající technologii webových služeb poskytuje funkcionalitu pro vygenerování zástupného objektu (tzv. proxy) z WSDL. Také proto se webové služby velice často vyvíjí stylem „contract-first“, tzn. nejdříve se obě strany dohodnou

[11] více informací o Informačním systému datových schránek lze nalézt na URL `http://www.datoveschranky.info/` (leden 2010).

na WSDL a poté si z tohoto platformě-nezávislého popisu každý vygeneruje svou část (webovou službu či proxy pro klienta).

WSDL v.1.1 z roku 2001 je dnes nejpoužívanější verze WSDL. Později vznikla verze WSDL 2.0, nicméně ta se zatím plně neujala a samotné konsorcium W3C tuto verzi označilo za W3C Candidate Recommendation.

Přesnou specifikaci WSDL 1.1 lze nalézt na webových stránkách konsorcia W3C (viz [5]).

4.3 Seznam dostupných webových služeb (WS-Inspection)

Web Service Inspection Language (WS-Inspection) je specifikace pro popis XML dokumentu, který reprezentuje seznam dostupných webových služeb, včetně URL adresy pro stažení WSDL definice webové služby. Tento seznam webových služeb by měl být dostupný jakou soubor s názvem *inspection.wsil* v kořenovém adresáři webu. Specifikace WS-Inspection 1.0[6] umožňuje umístit do tohoto souboru mnoho užitečných informací, jako např. textový popis služby, odkaz do UDDI registru, apod. Základní podoba *inspection.wsil* souboru je zobrazena na výpisu kódu 6.

```
1 <inspection:wsil xmlns:inspection="http://schemas.xmlsoap.org/ws/2001/10/inspection/">
2     <service>
3         <name>webserviceName</name>
4         <description location="http://server:port/webserviceName?wsdl" referencedNamespace="
              http://schemas.xmlsoap.org/wsdl/"/>
5     </service>
6 </inspection:wsil>
```

Výpis 6: Ukázka WSIL souboru

5 Předmět implementace

Multiagentní systém je možno definovat jako skupinu agentů, kteří jsou schopni jednat a provádět operace nezávisle, případně na základě koordinace s ostatními agenty. Komunikace mezi agenty probíhá pomocí jazyka KQML, případně ACL[12]. Multiagentní systém je tedy množina agentů, kteří provádějí operace a komunikují mezi sebou. Chceme-li kontaktovat konkrétního agenta A, musíme vytvořit agenta B, který pošle zprávu agentovi A. Propojení takového systému s jinými aplikacemi vyžaduje znalost problematiky MAS, jazyka ACL (případně KQML) a v neposlední řadě také spoustu úsilí.

Jinými slovy, chceme-li propojit MAS s jinými systémy, je vhodné navrhnout obecně použitelné řešení, tzv. bránu mezi prostředím MAS a ostatními aplikacemi. Brána musí být schopna přijmout požadavek od kterékoliv aplikace

1. nezávisle na platformě, na které aplikace běží (Windows, Linux, apod.),
2. nezávisle na jazyku, ve kterém je aplikace naprogramována (Java, C++, Perl, PHP, apod.)

a kontaktovat konkrétního agenta.

5.1 Komunikační brána

Jelikož je nutné dodržet požadavek nezávislosti brány na platformě a jazyku aplikace, která bude bránu používat, je vhodné použít technologii webových služeb. Brána se tedy bude chovat jako webová služba. Komunikace s webovou službou probíhá pomocí XML zpráv definovanými protokolem SOAP. Na úrovni transportní vrstvy je obvykle použit protokol HTTP, který je dostupný prakticky ze všech platforem a jazyků.

5.2 Komunikace brány a multi-agentního systému

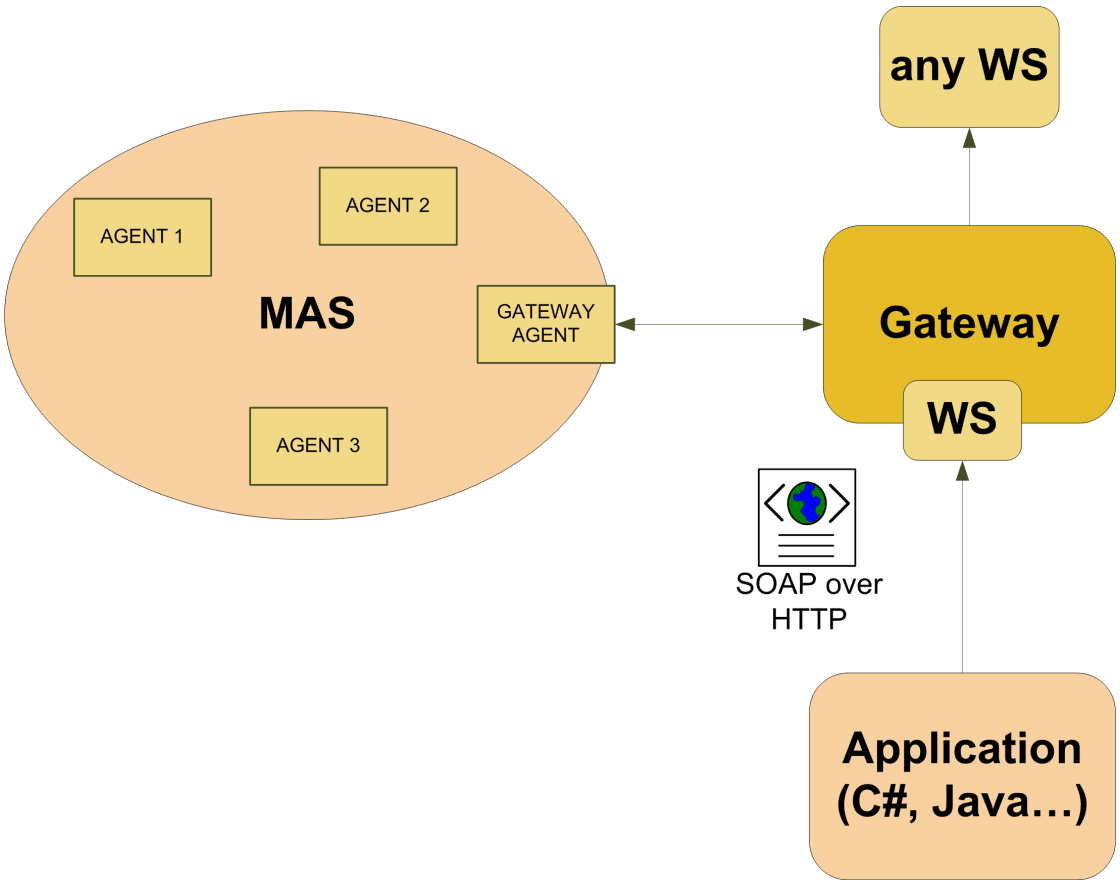
Brána tedy příjme SOAP zprávu s informacemi o volaném agentovi, názvem volané operace a případně i s parametry volání. Tyto informace musí předat dál do MAS (respektive konkrétnímu agentovi).

A jak již bylo řečeno v kapitole 5, je vhodné vytvořit zvláštního agenta (říkejme mu Gateway Agent), který bude schopen komunikovat se všemi ostatními agenty v MAS a zároveň bude schopen komunikace s naší bránou.

Brána tedy předá všechny přijaté požadavky volání Gateway Agentovi, který je pošle cílovému agentovi v MAS. Cílový agent provede danou operaci a vrátí výsledek Gateway Agentovi. Gateway Agent předá tento výsledek bráně, která jej ve formě SOAP zprávy vrátí klientovi webové služby (tzn.iniciátorovi celého procesu).

Na obrázku 5 je znázorněn koncept výše popsané brány včetně komunikačních kanálů.

[12] Rozdíly mezi KQML a ACL jsou popsané např. v publikaci [7]



Obrázek 5: Koncept brány MAS-WS

6 Specifikace požadavků

Cílem této kapitoly je shromáždit a popsat všechny požadavky na systém. Tyto požadavky budou dále použity jako vstup do procesu analýza řešení, viz následující kapitola 7.

Navržené řešení musí splňovat řadu kritérií a požadavků, které je možno rozdělit na **funkční** a **nefunkční**.

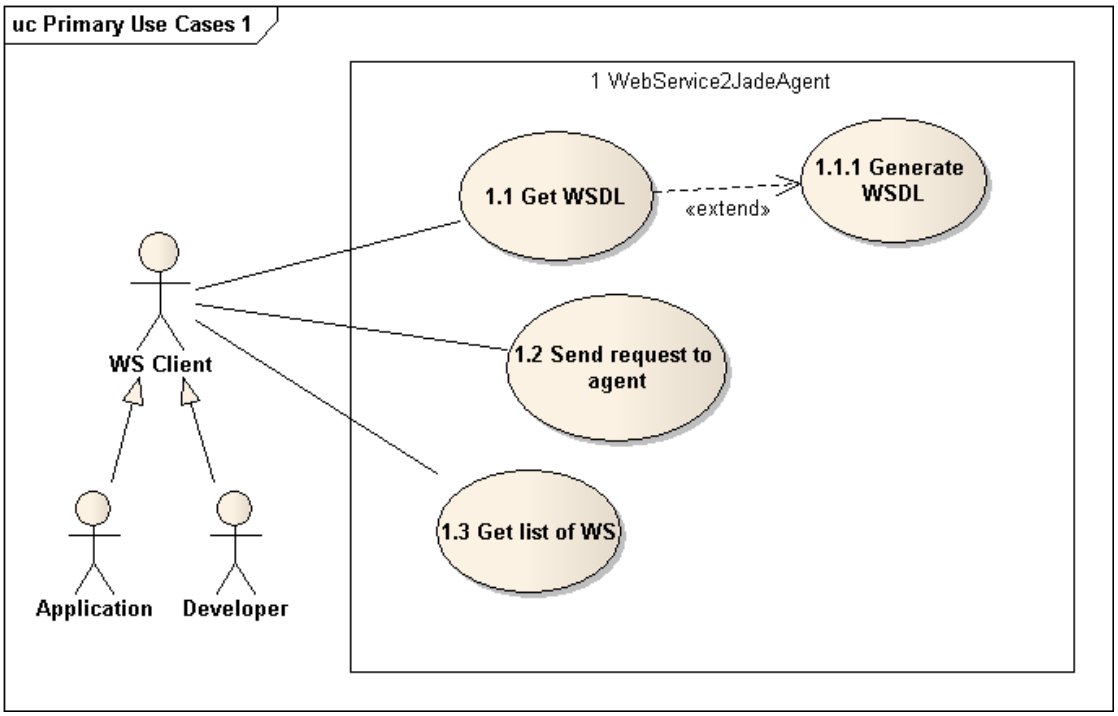
6.1 Funkční požadavky

Funkční požadavky tvoří základní stavební kámen specifikace aplikace. Funkčními požadavky rozumíme konkrétní chování aplikace (případy užití). Pro specifikaci funkčních požadavků se obvykle používají tzv. use case diagramy doplněné slovním popisem.

Požadavkem na aplikaci je umožnit dva způsoby komunikace:

1. komunikace: klient webové služby - webová služba - Agent,
2. komunikace: agent - webová služba.

Uvedené 2 způsoby komunikace tvoří 2 hlavní scénáře a jsou dále použity pro popis jednotlivých případů užití.



Obrázek 6: Případy užití pro komunikaci klient - webová služba - agent

6.1.1 Případ užití: Získat WSDL

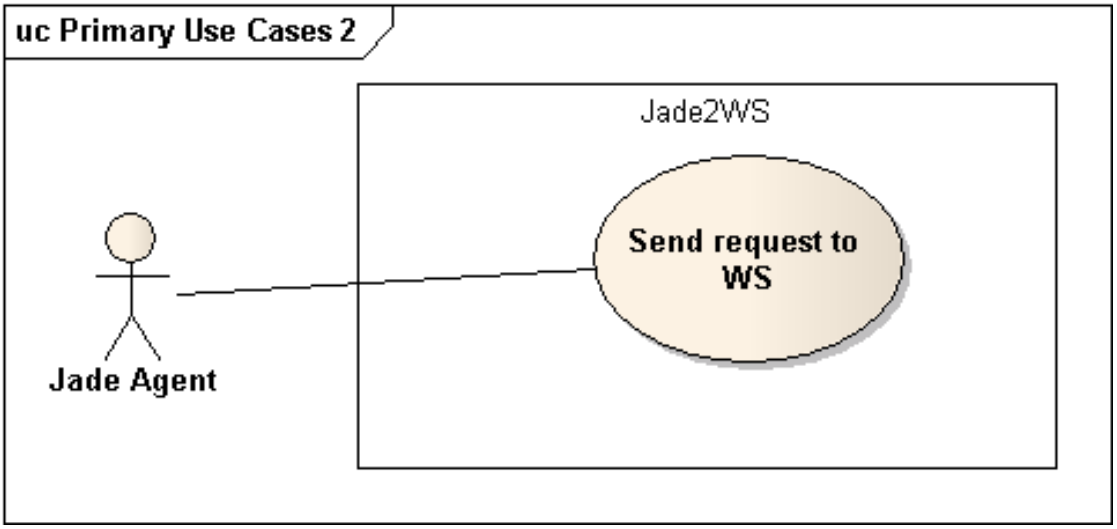
Identifikátor:	1.1
Popis:	Klient webové služby chce získat popis rozhraní webové služby (WSDL).
Aktéři:	Klient webové služby
Vstupní podmínky:	Webová služba je dostupná
Alternativní scénáře:	1.1.1 Generování WSDL
Hlavní scénář:	1. [*klient*] vyšle požadavek na webovou službu ve tvaru *http://host:port/agentName.agent?WSDL*, 2. [*systém*] KDYŽ WSDL definice existuje, TAK načte požadované WSDL 2.1 [*systém*] JINAK Alternativní scénář 1.1.1 Generovat WSDL 3. [*systém*] vrátí klientovi WSDL definici webové služby,
Výstupní podmínky:	Klient webové služby získal WSDL

6.1.2 Případ užití: Generovat WSDL

Identifikátor:	1.1.1
Popis:	Klient webové služby chce získat popis rozhraní webové služby (WSDL), které zatím není dostupné (není vygenerované).
Aktéři:	Webová služba
Vstupní podmínky:	Agent reprezentující webovou službu, je dostupný v prostředí MAS (jinak je automaticky zobrazena chybová zpráva)
Alternativní scénáře:	žádné
Hlavní scénář:	1. [*systém*] získá ontologii, 2. [*systém*] vygeneruje WSDL.
Výstupní podmínky:	Bylo vygenerováno WSDL

6.1.3 Případ užití: Poslání požadavku agentovi

Identifikátor:	1.2
Popis:	Klient webové služby chce poslat zprávu konkrétnímu agentovi do prostředí MAS.
Aktéři:	Klient webové služby, Webová služba, Agent
Vstupní podmínky:	• Agent reprezentující webovou službu, je dostupný v prostředí MAS, • Agent má dovoleno přijímat požadavky přes webovou službu, • Klient webové služby implementuje rozhraní webové služby, • Webová služba je dostupná,
Alternativní scénáře:	žádné
Hlavní scénář:	1. [*klient*] pošle požadavek na webovou službu, 2. [*webová služba*] předá požadavek agentovi, 3. [*agent*] zpracuje požadavek, 4. [*agent*] vrátí odpověď webové službě, 5. [*webová služba*] vrátí odpověď klientovi,
Výstupní podmínky:	Klient webové služby dostal od agenta odpověď na jeho požadavek.



Obrázek 7: Případy užití pro komunikaci agent - klient webové služby - webová služba

6.1.4 Případ užití: Poslat požadavek na webovou službu

Identifikátor:	2.1
Popis:	Agent Jade chce poslat požadavek na webovou službu.
Aktéři:	• Agent Jade, • Klient webové služby, • Webová služba.
Vstupní podmínky:	• Agent zná ontologii pro komunikaci s webovou službou, • Webová služba je dostupná.
Alternativní scénáře:	žádné
Hlavní scénář:	1. [*agent*] pošle požadavek klientovi webové služby, 2. [*klient webové služby*] předá požadavek na webovou službu, 3. [*webová služba*] zpracuje a vrátí výsledek klientovi, 4. [*klient webové služby*] předá odpověď agentovi.
Výstupní podmínky:	Agent získal odpověď od WS na svůj požadavek.

6.2 Nefunkční požadavky

Nefunkční požadavky obvykle charakterizují vlastnosti požadované aplikace jako např. použitá technologie, očekávané zatížení, dostupnost, bezpečnost, apod.

Hlavním nefunkčním požadavkem je použití implementačního rámce s názvem JadeLEAP. Implementace brány mezi MAS a WS má být provedena ve frameworku .NET (verze 3.5), konkrétně v jazyku C#.

Jak již z názvu této práce vyplývá, navržené řešení musí být postavené na technologii webových služeb.

7 Analýza

V rámci analýzy řešení byly důkladně prověřeny možnosti frameworku Jade s ohledem na komunikaci agentů, přijímání požadavků z vnějšku (mimo Jade kontejner), apod. Rovněž nebyla opomenuta analýza existujícího řešení, viz kapitola 7.1. Výsledky všech těchto aktivit budou vstupem do procesu návrhu řešení, viz kapitola 8.

Součástí analýzy řešení je rovněž identifikace možností implementace v .NET frameworku, neboť použití .NET frameworku je jeden z klíčových požadavků na výsledné řešení.

7.1 Analýza existujícího řešení WSIG (Java)

Pro jazyk Java existuje projekt s názvem Web Service Integration Gateway (WSIG)[13], který poskytuje řešení pro komunikaci s Multi-Agentním systémem prostřednictvím webových služeb. Za vývojem WSIG stojí švýcarská společnost Whitestein, která je účastníkem pracovní skupiny pro výzkum propojení agentů a webových služeb (Agent and Web Service Interoperability).

Projekt WSIG je postaven na frameworku Jade.

7.1.1 Popis řešení WSIG

Hlavní částí projektu WSIG je servlet, který musí být spuštěn v servletovém kontejneru nebo aplikačním serveru (Tomcat, JBoss, Glassfish, atd.). Při inicializaci tohoto servletu je vytvořen Agent kontejner, který se připojí k hlavnímu kontejneru (dle konfigurace). Po úspěšném připojení je vytvořen nový agent Gateway Agent, který je schopen přijímat přihlášky (subscriptions) o registraci nových agentů do DF. Je-li zaregistrován nový agent, Gateway Agent obdrží informaci o registrovaném agentu a podnikne tyto kroky:

- zaregistruje ontologii nového agenta mezi známé ontologie,
- vygeneruje WSDL popis webové služby reprezentující nového agenta,
- vytvoří instanci třídy WSIGService (a nastaví všechny informace o službě - název, prefix, wsdl, AID agenta),
- vytvořenou instanci třídy WSIGService přidá do uložiště *WSIGStore*, které je umístěno v kontextu daného servletu.

Tímto je proces vytvoření webové služby pro nového agenta ukončen. Pokud klient webové služby vyžaduje WSDL definici konkrétní webové služby(agenta), je nutné zavolat servlet a potažmo celý servletový kontejner s URL obsahující proměnnou WSDL, např. http://localhost:8080/wsig/ws/MatchService?WSDL. *WSIGServlet* podle názvu webové služby (v našem případě je to „MatchService“) vyhledá v uložišti zaregistrovaných webových služeb (WSIGStore) konkrétní instanci *WSIGService* a vrátí klientovi WSDL definici. Jednotlivé kroky při získávání WSDL definice WS jsou znázorněný sekvenčním diagramem na obrázku 8.

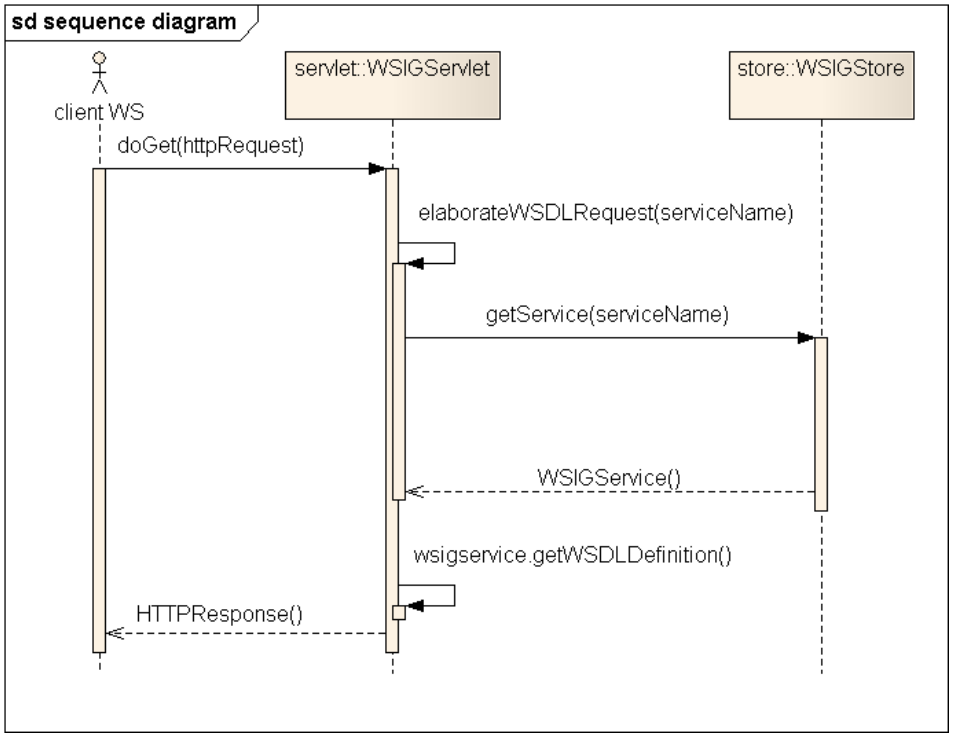
[13] Aktuální verze je 2.0 (říjen 2008) a je ke stažení na adrese `http://jade.tilab.com/dl.php?file=wsigAddOn-2.0.zip`

7.1.2 Komunikace mezi klientem WS a agentem

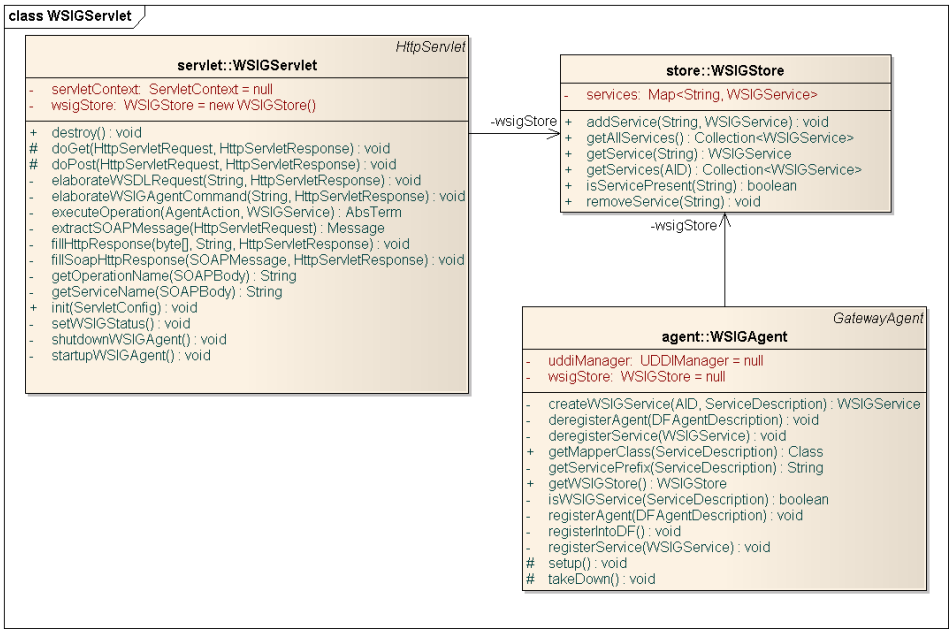
V projektu WSIG není každý agent zastoupen webovou službou jako takovou, ale jedním servletem, který přijímá SOAP požadavky. To znamená, že chce-li klient zavolat přes webovou službu konkrétního agenta, pošle svůj SOAP požadavek na adresu *WSIGServletu*. Tento servlet zpracuje přijatou SOAP zprávu, ze které zjistí:

- název webové služby, která je volána,
- metoda webové služby, která je volána,
- parametry volané metody.

Na základě těchto informací vyhledá v uložišti *WSIGStore* konkrétní instanci *WSIGService* reprezentující danou WS. Poté je vytvořen nová instance *Behaviour* s informacemi o *AID* příjemce a akci, která má být vykonána. Toto chování je předáno agentovi GatewayAgent k provedení. GatewayAgent pošle příjemci *ACL* zprávu (dle konfigurace chování) a obdrženou odpověď vrátí zpět servletu. Servlet obalí odpověď do SOAP zprávy a vrátí ji klientovi.



Obrázek 8: Sekvenční diagram požadavku na WSDL definici WS



Obrázek 9: Třídní diagram servletu WSIGServlet

7.1.3 Pozitiva a negativa projektu WSIG

WSIG je hotové řešení, které funguje. Proto je vhodné poučit se z chyb a nedokonalostí, které obsahuje. A také naopak - není na škodu některé prvky z architektury WSIG převzít a dále je rozvinout.

Výhody a dobré vlastnosti architektury projektu WSIG:

- použití obecné komponenty (určené k HTTP komunikaci), která je díky svým vlastnostem schopna spravovat veškerou komunikaci, viz kapitola 7.1.5 Servlet,
- využití jednotného uložiště pro sdílení informací pro aplikaci, viz kapitola 7.1.6 Servlet Context.

Nevýhody a nedostatky architektury projektu WSIG:

- projekt neumožňuje komunikaci směrem od agenta na webovou službu,
- je závislý na verzi frameworku Jade (podporována pouze verze 3.5 a výše, kde už je pro toto řešení připravena podpora ve formě nových metod[14], atd.),
- je dostupný pouze pro platformu Java.

[14]WSIG aktivně využívá třídu JadeGateway, která je dostupná až od verze 3.5, více info o této třídě viz `http://jade.tilab.com/doc/api/jade/wrapper/gateway/JadeGateway.html` (říjen 2009)

7.1.4 Možnosti využití výhod projektu WSIG na platformě .NET

Jak již bylo řečeno, projekt WSIG je napsán v programovacím jazyce Java. Platforma .NET je od platformy Java hodně odlišná. Prvky jako servlet, kontext servletu, apod. se na platformě .NET vůbec nevyskytují, proto se musíme zamyslet nad jejich významem v projektu WSIG a pokusit se nalézt řešení z platformy .NET, které by nám tyto prvky plně nahradily.

7.1.5 Servlet

Servlet je programová komponenta běžící na straně serveru. Dle specifikace je servlet nevizuální aplikace, která přijímá požadavek a na jejím základě generuje odpověd [11]. Komunikace se servletem probíhá nejčastěji pomocí protokolu HTTP. Obvyklou formou odpovědi je HTML kód (reprezentující HTML stránku, kterou klient požadoval).

V projektu WSIG má servlet tyto funkce:

- přijímá a zpracovává SOAP zprávy,
- přijímá požadavky na WSDL definici WS (tento WSDL je dále vrácen klientovi),
- po dobu běhu servletu uchovává instance tříd definující rozhraní WS pro jednotlivé agenty (k této funkcionalitě je použit ServletContext).

Z platformy .NET se servletu nejvíce podobá (funkčně) *HTTPHandler*. *HTTPHandler* je funkční modul, který zpracovává požadavky jež jsou cílené na konkrétní adresu[12]. Každý *HTTPHandler* musí implementovat interface *IHTTPHandler* a překrýt metodu *ProcessRequest(HttpContext context)* a vlastnost *IsReusable*. Metoda *ProcessRequest* má za úkol zpracovat požadavek (request) a vrátit na výstup odpověd (HTML stránku, binární data, apod.). Vlastnost *isReusable* vrací booleovskou hodnotu (true nebo false) podle toho, zda může další požadavek využívat stejnou instanci *HttpHandleru* či nikoliv. Jednoduchý *HTTPHandler* je znázorněn na výpisu zdrojového kódu č. 7.

```
1  using System.Web;
2  public class HelloWorldHandler : IHttpHandler
3  {
4      public HelloWorldHandler()
5      {     }
6      public void ProcessRequest(HttpContext context)
7      {
8          HttpResponse Response = context.Response;
9          Response.Write("<html>");
10         Response.Write("<body>");
11         Response.Write("<h1>Hello from a˜custom HTTP handler.</h1>");
12         Response.Write("</body>");
13         Response.Write("</html>");
14     }
15     public bool IsReusable
16     {
17         get { return false; }
18     }
19 }
```

Výpis 7: Jednoduchý HTTPHandler

7.1.6 ServletContext

ServletContext je interface, díky kterému se může servlet dostat k informacím o prostředí, ve kterém je spuštěn. Jako například inicializační parametry pro servletový kontejner, ve kterém servlet běží. *ServletContext* je dostupný ze všech částí běžící aplikace. Tato vlastnost napomáhá využití ServletContextu k uchování objektů, které jsou dostupné v rámci celé aplikace (např. spojení na databázi, logger, atd.). A právě tato funkcionalita je využita v projektu WSIG ke zpřístupnění instance objektu *WSIGStore* v rámci celého projektu. (*WSIGStore* je uložiště definic webových služeb jednotlivých agentů).

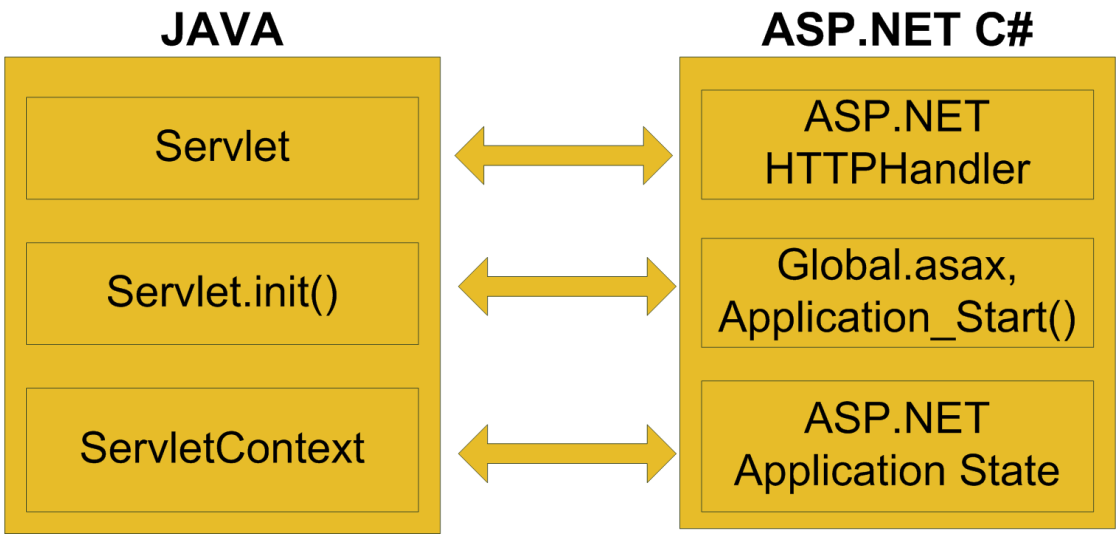
Obdobnou funkcionalitu lze na platformě .NET nalézt pod označením *ASP.NET Application State*. *Application State* je datové uložiště dostupné pro všechny třídy v ASP.NET aplikaci. Data, která jsou uložena do *Application State*, jsou uchována v paměti serveru a práce s nimi je rychlejší než práce s daty v databázi. Nicméně v případě restartu či výpadku serveru jsou nenávratně ztracena. [13]

7.1.7 Inicializace Servletu

Aplikace WSIG při svém startu spouští Agent kontejner (s Gateway Agentem), který se připojí k hlavnímu kontejneru. Toto spuštění je nutné provést pouze jednou a proto je tato aktivita součástí servletu a jeho metody *init()*. Pro metodu *init()* je specifické to, že je zavolána jen a pouze jednou [11].

Podobnou charakteristiku jako metoda *init()* má na platformě .NET modul, který je známý pod označením *Global.asax*, respektive jeho metoda *Application_Start()*. Jediný rozdíl tkví v mechanismu, který tuto metodu spustí. Zatímco v javě je metoda *init()* zavo-

lána automaticky při spouštění aplikačního serveru, metoda *Application_start()* je zavolána až při prvním požadavku na aplikaci.

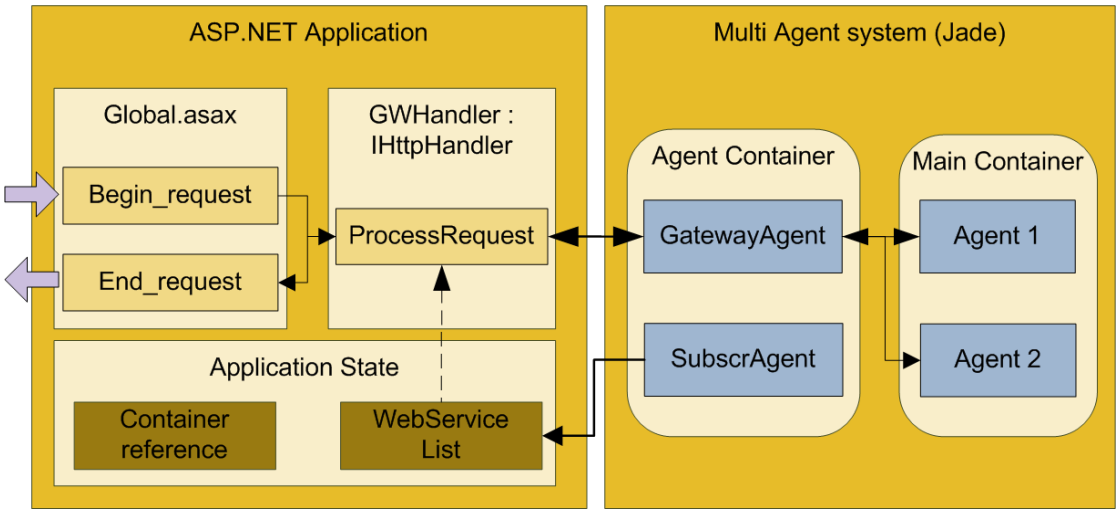


Obrázek 10: Hlavní prvky WSIG(Java) a jejich ekvivalence na platformě .NET

8 Návrh

Při návrhu řešení je nutné vycházet z výsledků analýzy řešení, proto je vhodné si zopakovat nejdůležitější výsledky analýzy.

V kapitole 7.1.4 byly popsány tři hlavní programové moduly navrhovaného řešení. Jedná se o *HTTPHandler*, *ASP.NET Application State* a *Global.asax*. Dalším vstupem pro návrh řešení je způsob, jakým lze komunikovat s agenty z vnějšku Multi-agentního systému. V kapitole 5.2 bylo zmíněno, že k interakci s jednotlivými agenty je vhodné vytvořit zvláštního agenta (Gateway Agent), kterému budeme předávat požadavky. Tento základní pohled na celou architekturu je znázorněn na obrázku 11.



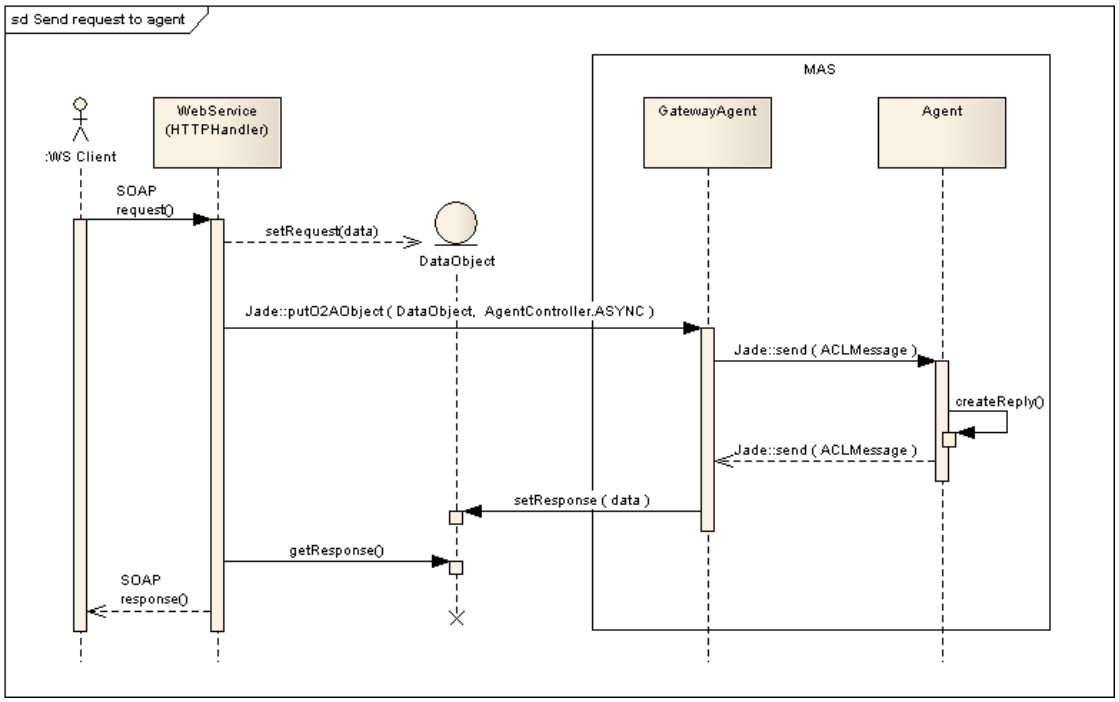
Obrázek 11: Obecný pohled na architekturu

8.1 Komunikace WS - Gateway Agent - Agent

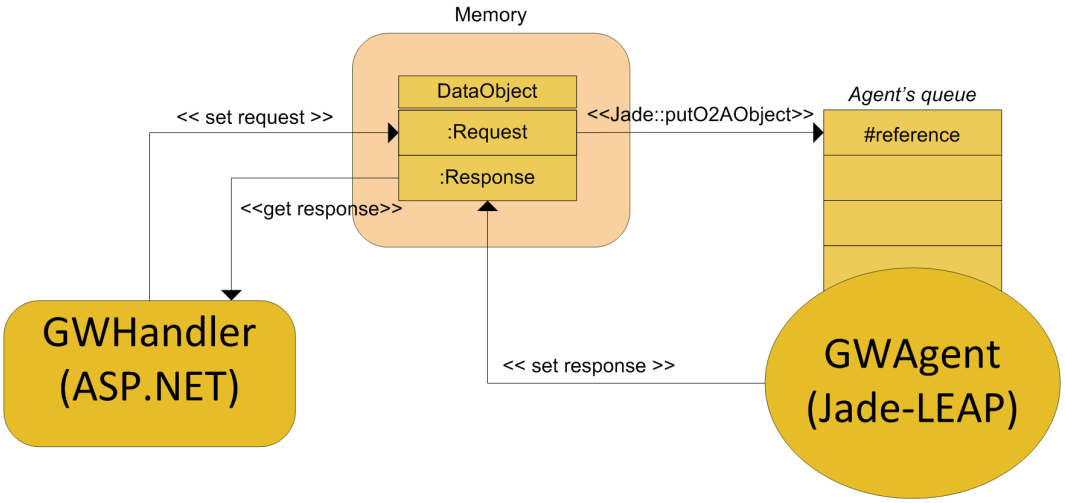
Klient webové služby pošle webové službě požadavek ve formátu SOAP. Tato webová služba požadavek zpracuje a předá Gateway Agentovi pomocí metody *putO2AObject(Object o, bool blocking)*. Volání této metody je asynchronní[15] a nemá žádnou návratovou hodnotu (void). My ale potřebujeme získat odpověď na naše volání. Tento nedostatek frameworku Jade lze vyřešit tak, že Gateway Agentu předáme referenci na instanci třídy DataObject, což je vlastní datová struktura pro předávání požadavků a odpovědí mezi agentem a webovou službou, viz obrázek 14.

Po předání této reference Gateway Agentovi se vlákno webové služby zastaví a bude čekat na uvolnění zámku nad objektem třídy *DataObject*.

[15] metoda *putO2AObject(Object o, bool blocking)* má dva parametry, prvním je samotný objekt, druhým parametrem je boolean hodnota, která říká, že další zpracování má či nemá počkat do doby, než si agent vyzvedne Object o ze své fronty požadavků. Tzn. pomocí tohoto parametru nelze říci, že vlákno bude čekat do doby než agent zpracuje požadavek a vrátí odpověď.



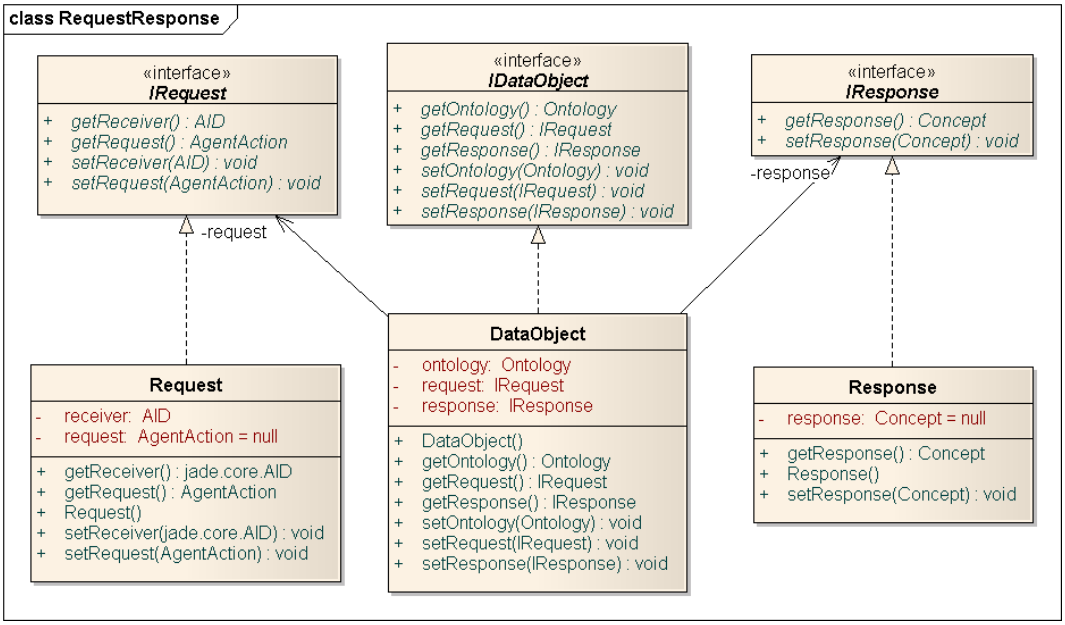
Obrázek 12: Sekvenční diagram - komunikace WS client to agent



Obrázek 13: Způsob komunikace prostřednictvím přístupu ke sdílené datové struktuře DataObject

Během této doby odešle Gateway Agent požadavek konkrétnímu agentovi, který jej zpracuje a pošle zpět odpověď. Gateway Agent odpovědí naplní instanční proměnnou typu *Response* z objektu třídy *DataObject* a oznámí uvolnění zámku nad tímto objektem.

V tuto chvíli se probudí vlákno webové služby, přečte odpověď z objektu třídy *DataObject* a předá ji klientovi webové služby. Všechny tyto aktivity jsou znázorněny v sekvenčním diagramu na obrázku 12. Třídní diagram datových struktur použitých v této komunikaci je na obrázku 14.



Obrázek 14: Třídní diagram - Datové struktury pro komunikaci s agenty

8.2 Komunikace Agent - WS Invoker Agent - Webová služba

Tento způsob komunikace z MAS směrem na webové služby vyžaduje implementaci nového agenta, který bude simulovat webové služby. Všechny požadavky z MAS, které budou adresovány webovým službám na internetu, budou ve skutečnosti směrovány tomuto agentovi, který požadavek zpracuje, zavolá konkrétní webovou službu a vrátí výsledek.

Implementace WS Invoker Agenta je založena na principu tzv. dynamického volání webových služeb. V požadavku, který WS Invoker agent příjme, musí být tyto informace:

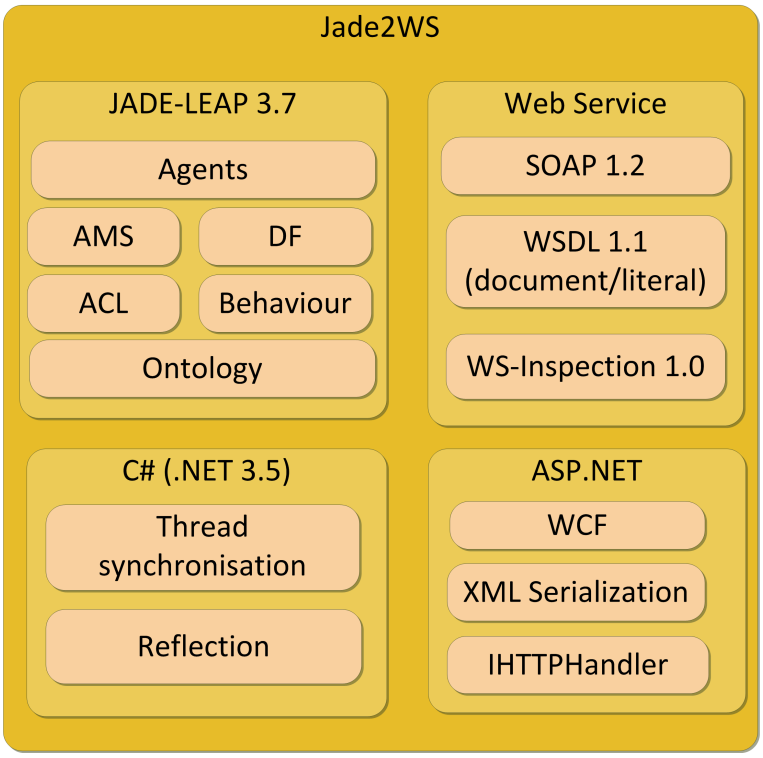
- URL adresa, kde je možno stáhnout WSDL definici webové služby,
- název webové služby shodný s pojmenováním webové služby ve WSDL,
- název volané metody,
- pole objektů = parametry volání dané metody.

Tento agent si sám stáhne WSDL ze zadané URL adresy, vygeneruje si zástupný objekt (proxy). Následně se pomoci reflexe zavolá konkrétní metoda s polem parametrů. Na pozadí se provede zavolání dané webové služby. Výsledek celého zpracování je pak vrácen původnímu agentovi, který žádal o zavolání webové služby.

9 Implementace

9.1 Použité technologie

Pro implementaci celého řešení brány mezi MAS a WS bylo použito mnoho různých technologií, např. ASP.NET, framework Jade-LEAP, atd. Seznam všech použitých technologií lze nalézt na obrázku 15.



Obrázek 15: Použité technologie

9.2 Rozdělení aplikace

Celé řešení brány mezi MAS a WS tvoří jedna ASP.NET aplikace skládající se ze dvou částí (toto rozdělení vyplývá z návrhu řešení, viz kapitola 8). První část tvoří dvě komponenty sloužící pro komunikaci s agenty, respektive pro získání WSDL definice konkrétního agenta. Druhou část aplikace tvoří speciální agenti určení pro podporu celého řešení zevnitř MAS (JADE).

Jednou z dvou komponent tvořící první část řešení je komponenta *GatewayHandler*, která zpracovává příchozí SOAP požadavky a předává je dále ke zpracování Gateway Agentovi. Druhým prvkem této ASP.NET části je *WSDLHandler*, který zpracovává požadavky na WSDL definice agentů, respektive webových služeb zastupujících agenty. Obě

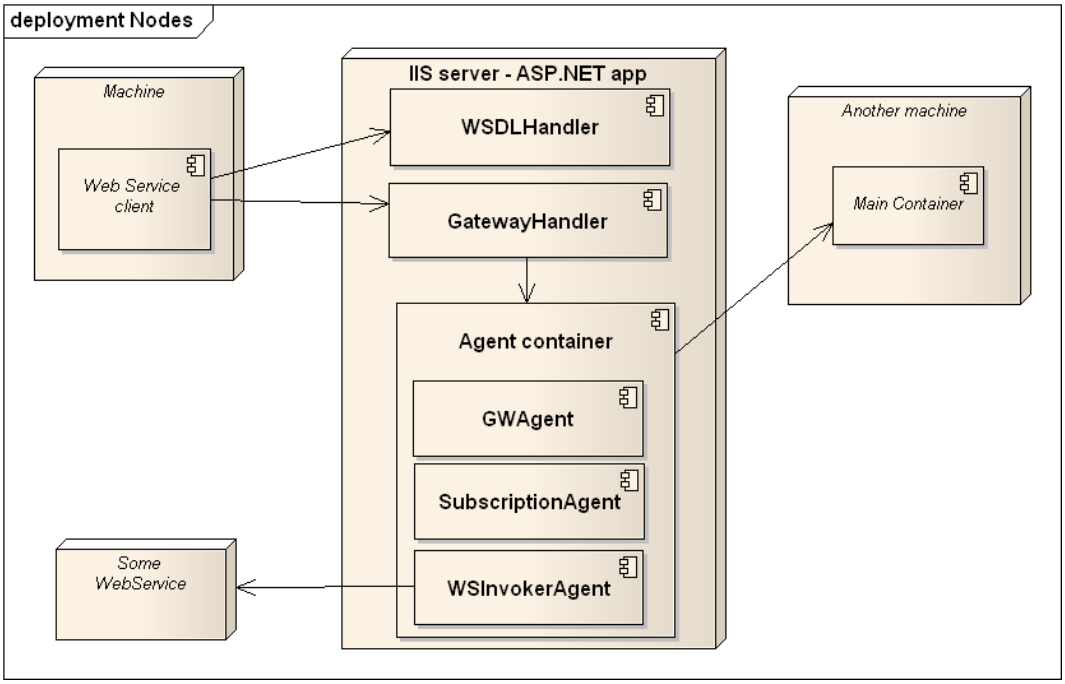
tyto komponenty jsou vytvořeny jako *HTTPHandler* (implementují interface *IHTTPHandler*).

Druhá část aplikace je tvořena z těchto agentů:

1. Gateway Agent,
2. WSInvoker Agent,
3. Subscription Agent.

Hlavním úkolem Gateway Agenta je přijímat požadavky od komponenty GatewayHandler a přeposílat je konkrétním agentům v MAS. Více informací lze nalézt v kapitole 9.5. WSInvoker Agent je schopen přijímat požadavky na konkrétní webové služby a ty pak provést, včetně vrácení výsledku konkrétnímu agentovi. A konečně Subscription Agent naslouchá registraci nových agentů a pro každého nového agenta vytvoří WSDL a zaregistruje jeho iniciály do interního uložiště sdíleného s komponentu GatewayHandler.

Rozmístění jednotlivých komponent aplikace je zobrazeno na diagramu nasazení, viz obrázek 16.



Obrázek 16: Rozmístění jednotlivých komponent aplikace

9.3 Inicializace aplikace

Při prvním HTTP požadavku na tuto ASP.NET aplikaci je vytvořen Agent kontejner, ve kterém jsou spuštěni následující agenti:

- Gateway Agent, viz kapitola 9.5,
- Subscription Agent, viz kapitola 9.9,
- WSInvoker Agent, viz kapitola 9.6.

Celý tento Agent kontejner je následně připojen k hlavnímu kontejneru a plní funkci brány mezi MAS a webovými službami. Celý proces inicializace a zpracování SOAP požadavku je znázorněn na aktivitním diagramu, viz obrázek 17.

Pro demonstraci celé aplikace je nutné spustit hlavní kontejner, který tvoří centrální bod Jade platformy a musí být spuštěn jako první. Při reálném použití je samozřejmě možné Agent kontejner připojit k jakémukoliv jinému hlavnímu kontejneru.

Mezi hlavní výhodu spouštění Agent kontejneru na straně ASP.NET aplikace patří nezávislost již existujících hlavních a Agent kontejnerů na této WebGateway, tzn. WebGateway je možno připojit či odpojit za běhu celého MAS (Jade). Tato vlastnost je neocenitelná obzvláště v případě, kdy celé MAS tvoří několik desítek Agent kontejnerů, které jsou spuštěny na různých platformách (PC, mobilní zařízení, apod.), a jakýkoliv pokus o restart nebo přenastavení všech těchto kontejnerů je v podstatě vyloučen.

9.4 Vstupní bod aplikace - GatewayHandler

Centrálním prvkem celého řešení je vlastní *HTTPHandler* s názvem GatewayHandler. Tento modul tvoří jednu ze dvou částí komunikační brány mezi světem MAS a okolím (druhou částí je Gateway Agent, viz kapitola 9.5). Jeho úkolem je přijmout a zpracovat všechny HTTP požadavky, které směřují na URL ve tvaru `http://server:port/nazevAgenta.agent`. Komunikace se všemi agenty tedy probíhá přes tento jediný prvek - GatewayHandler. Použití jediného prvku zpracovávajícího požadavky na všechny agenty klade nároky na výkon použitých mechanismů a technologií (obzvláště při vyšším zatížení). Nejen proto je čtenář v kapitole 10 seznámen s výsledky výkonnostních testů celé aplikace.

Každý HTTP požadavek, který má GatewayHandler zpracovat, musí obsahovat volání agenta reprezentovaného patřičnou SOAP zprávou (SOAP v.1.2). GatewayHandler všechny tyto volání předá cílovému agentovi prostřednictvím prostředníka - Gateway Agenta.

Neméně důležitým úkolem GatewayHandlera je tzv.error handling. V případě, že dojde při zpracovávání SOAP požadavku k nějaké neočekávané události (výjimce), je povinnosti GatewayHandlera patřičně notifikovat klienta webové služby o této události. K notifikování výjimek je použit systém tzv. SOAP Faultů, což je obdoba výjimek(exceptions) z dnešních programovacích jazyků.

GatewayHandler používá pro zpracování příchozích a odchozích SOAP zpráv technologii XML serializace, která je dále popsána v kapitole 9.7.

9.5 Komunikační prostředník č.1 - Gateway Agent

Jak již bylo řečeno v kapitole 5.2, pro přeposílání komunikace z vnějšího světa MAS k jednotlivým agentům, je vytvořen nový agent s názvem Gateway Agent. Tento agent pravidelně kontroluje svou frontu požadavků a jakmile se v ní objeví objekt typu DataObject, vytvoří příslušnou ACLMessage a odešle požadavek cílovému agentovi. Poté počká na odpověď a až poté uvolní objekt typu DataObject k dalšímu zpracování GatewayHandlerem. Celý tento proces je znázorněn na obrázku 12.

9.6 Komunikační prostředník č.2 - WSInvoker Agent

WS Invoker agent je agent zastupující webové služby na internetu. Je vytvořen tak, aby umožnil agentům komunikaci s webovými službami jen na základě množiny informací {*URL adresa, název služby, název metody, pole parametrů*}. O vše ostatní se již postará tento WS Invoker Agent.

Pro komunikaci s tímto Agentem je vytvořena speciální ontologie, která je definována takto:

- AgentActionSchema WS Request
  - SERVICE URL - URL adresa směřující na WSDL definici webové služby,
  - SERVICE NAME - název webové služby shodný s pojmenováním služby ve WSDL,
  - METHOD NAME - název metody shodný s názvem metody ve WSDL,
  - ARGUMENTS - pole argumentů jakéhokoliv typu.
- ConceptSchema - WS Response
  - WS RESPONSE OBJECT - objekt reprezentující odpověď.

Pokud WS Invoker Agent obdrží zprávu definovanou výše uvedenou ontologií, stáhne si z uvedené URL adresy WSDL definici webové služby a vygeneruje si zástupný objekt - proxy. Na pozadí tohoto procesu se vytvoří DLL soubor s touto proxy a uloží se do dočasného uložiště dle konfigurace Windows[16]. Poté co je zástupný objekt vytvořen, je pomocí reflexe zavolána konkrétní metoda, které jsou zároveň předány všechny parametry. Výsledek volání je pak vrácen zpět agentovi, který žádal o zavolání webové služby.

Poznámka 9.1 Aktuální implementace obsahuje omezení, které dovoluje volat metody webové služby jen s použitím primitivních datových typů. Používání komplexních typů (complex-type) není podporováno. Důvodem tohoto omezení jsou problémy s vytvářením a přeposíláním instancí objektů těchto komplexních typů směrem od volajících agentů k WS Invoker Agentovi. Tento nedostatek může být v některé z dalších verzí odstraněn. Jednou z možných variant řešení je vytváření ontologii pro všechny webové služby, které jsou jednotlivými agenty volány. S touto variantou navíc souvisí nutnost vytvoření konfiguračního rozhraní pro správu všech zaregistrovaných webových služeb.

[16] Vytvořené DLL jsou obvykle ukládána do složky Documents and Settings/[user]/Local Settings/Temp

9.7 Zpracování příchozí SOAP zprávy

Komunikace mezi klientem webové služby a agentem (reprezentovaným webovou službou) probíhá na úrovní XML zpráv definovaných protokolem SOAP verze 1.2. Chce-li klient komunikovat s konkrétním agentem, musí poslat SOAP zprávu na URL adresu ve tvaru: *http://server:port/nazevAgenta.agent*
Z URL adresy tedy lze identifikovat název volaného agenta. Zbylé informace jako

- název volané metody,
- hodnoty parametrů této metody

jsou obsaženy v SOAP zprávě. Celý HTTP požadavek, včetně SOAP zprávy vypadá zhruba takto:

```
1  POST http://localhost:3643/Calculator.agent HTTP/1.1
2  Content-Type: application/soap+xml;charset=UTF-8;action="Agents.Sum"
3  Host: localhost:3643
4  Content-Length: 254
5
6  <soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope" xmlns:agen="
       Agents">
7      <soap:Header/>
8      <soap:Body>
9          <agen:Sum>
10             <agen:a>5</agen:a>
11             <agen:b>10</agen:b>
12         </agen:Sum>
13     </soap:Body>
14 </soap:Envelope>
```

Výpis 8: Ukázková SOAP zpráva volající metodu Sum s parametry 5 a 10.

Z hlaviček HTTP požadavku je důležité se zmínit o významu hlavičky *Content-type* a jejího atributu *action*[17]. Tento atribut určuje název volané operace(akce) a je použit pro mapování SOAP zprávy na daný objekt.

Máme-li název volané metody, pak druhým úkolem je namapovat SOAP zprávu na objekt dané třídy (v našem případě vytvoříme instanci třídy Sum). Jedna z variant je použít XML serializaci, respektive XML deserializaci. Definice 9.1 v podstatě říká, že XML serializace je pro nás ideálním řešením.

Definice 9.1 *„Primárním účelem XML serializace v .NET frameworku je umožnit konverzi XML dokumentů na runtime objekty a naopak“ [18].*

Podíváme-li se na možnosti serializace XML souborů na platformě .NET, zjistíme, že můžeme použít několik druhů tzv. formatterů, které dědí z interface IFormatter a umožňují ovlivnit zpracovávání serializovaných dat:

[17] Protokol SOAP v.1.1 tuto informaci reprezentuje trošičku jinak. Používá speciální hlavičku se jménem SOAPAction viz [17]

1. *BinaryFormatter* - umožňuje serializaci či deserializaci binárních dat, mimo jiné i XML,
2. *SOAPFormatter* - umožňuje serializaci či deserializaci SOAP zpráv,

SOAPFormatter se jeví jako nejlepší volba. Tato varianta skrývá jeden drobný problém. SoapFormatter je od verze .NET 3.5 označen jako zastaralý a již by se neměl dále používat [18].

BinaryFormatter je často označován jako ideální náhrada za zastaralý *SoapFormatter*. Ale ani tato varianta není nejvhodnější. *BinaryFormatter* při serializaci objektu do XML „obohatí“ výstupní XML o verzi použitého typu (+assembly) a tuto informaci využívá rovněž při deserializaci z XML do objektu. Důsledkem tohoto chování bývá často problém se zpracováním SOAP zprávy, která byla doručena bez poskytnutí těchto informací (které jsou specifické pouze pro .NET)[19]. Např. v případě, že SOAP zpráva byla vytvořena na jiné platformě než .NET.

Možná i díky výše uvedeným problémům se serializací SOAP zpráv přišel Microsoft s novým tzv.service-oriented formatterem, který nazval *DataContractSerializer*. Tento formatter je součástí *Windows Communication Foundation (WCF)*, což je SDK pro vývoj webových služeb na Windows. WCF bylo na trh uvolněno s frameworkem .NET verze 3.0 na sklonku roku 2006.

DataContractSerializer se vyznačuje vylepšenou podporou interoperability. Mimo jiné umožňuje velice jednoduše přepínat verzi protokolu SOAP použitou pro serializaci/deserializaci. *DataContractSerializer* je tedy jasnou volbou pro XML serializaci.

XML serializace v jazyce C# znamená provést tyto tři kroky:

1. přidat anotaci *[DataContract]* ke třídě, které má být využita pro serializaci a deserializaci, viz výpis zdrojového kódu 9
2. přidat anotaci *[DataMember]* ke členům této třídy, které mají být zohledněny při serializaci a deserializaci, viz výpis zdrojového kódu 9
3. provést samotnou serializaci a deserializaci za pomocí několika tříd jazyka C# jako DataContractSerializer, XmlDictionaryWriter, Message apod.

[18] Zmínka o zavrhnutí SoapFormatteru je mimo jiné zveřejněna v dokumentaci třídy SoapFormatter na stránce `http://msdn.microsoft.com/en-us/library/system.runtime.serialization.formatters.soap.soapformatter.aspx` (prosinec 2009)

```
1  [DataContract(Name = "Sum", Namespace = "Agents")]
2  public class Sum:AgentAction
3  {
4      [DataMember]
5      public int a;
6      [DataMember]
7      public int b;
8
9      public int getA()
10     {
11         return this.a;
12     }
13      ...
14  }
```

Výpis 9: Třída reprezentující operaci Sum obohacena o anotace pro snadnou serializaci

9.8 Vytvoření odchozí SOAP zprávy

Odchozí SOAP zpráva obsahuje odpověď volaného agenta s výsledkem požadované operace (metody). V rámci MAS je tato zpráva reprezentována jako objekt dané třídy. Má-li být odeslána klientovi webové služby, je nutné tuto zprávu převést do XML formátu odpovídající protokolu SOAP. K tomu použijeme osvědčenou XML serializaci, viz předchozí kapitola 9.7.

9.9 Registrace nových agentů

Každý agent musí být veřejně dostupný pomocí rozhraní webové služby. Rovněž pro každého agenta musí být veřejně dostupný popis jeho metod ve formátu WSDL. Abychom mohli výše uvedené umožnit, je nutné sbírat nutné informace o zaregistrovaných agentech. Nutnými informacemi se myslí:

- jméno agenta (AID) včetně názvu platformy, kde je tento agent dostupný,
- ontologie, která definuje datové struktury pro komunikaci s tímto agentem.

Framework Jade poskytuje komponentu *Directory Facilitator(DF)*, která poskytuje informace o zaregistrovaných agentech [8]. Služba, kterou tato komponenta poskytuje, se často nazývá yellow pages (zlaté stránky [16]). Každý agent se může(ale nemusí) přihlásit do této služby a poskytnout tak informace o sobě pro ostatní agenty. Komponenta DF samozřejmě poskytuje i možnost prohledávat tento registr.

Kromě výše uvedených funkci (registrace, deregistrace a vyhledávání agentů) poskytuje Jade mechanismus pro notifikovaní agentů o událostech jako registrace či deregistrace agenta, připojení či odpojení platformy, apod. Chce-li daný agent přijímat tyto informace, je nutné se přihlásit k odběru těchto informací. Jeden ze způsobů, jak lze toto provést, je uvedeném na výpisu kódu 10. Dalším způsob je popsán v publikaci „Developing Multi-Agent Systems with JADE“ [8], kapitola „5.5.2 Subscribing to platform events“.

```
class SubscriptionAgent:Agent
{
    public override void setup(){
        // create empty service description for this agent
        ServiceDescription sd = new ServiceDescription();

        DFAgentDescription tmplt = new DFAgentDescription();
        tmplt.addServices(sd);

        ACLMessage subscrMsg = null; // create subscription message
        subscrMsg = DFService.createSubscriptionMessage(this, getDefaultDF(), tmplt, null);

        // get wsStore forwarded to agent as an argument
        Hashtable wsStore = (Hashtable)getArguments()[0];

        // create instance of new behaviour
        // and forward reference to ws store object(hashtable) as an argument
        Behaviour subscrBhvr = new SubscriptionBehaviour(this, subscrMsg, wsStore)

        // add new behaviour that will takes care of subscriptions
        addBehaviour(subscrBhvr));
    }
}
```

Výpis 10: Přihlášení agenta k přijímání informací o registraci a deregistraci agentů

Chování, které bude mít na starosti zpracovávání informačních zpráv z yellow pages, bude dědit ze třídy *SubscriptionInitiator* a dále bude mít překrytou metodu *handleInform(ACLMessage inform)*. V této metodě se budou zpracovávat zprávy obsahující registrace/deregistrace jednotlivých agentů.

```
public override void handleInform(ACLMessage inform)
{
    DFAgentDescription[] description = DFService.decodeNotification(inform.getContent());
    foreach (DFAgentDescription desc in description)
    {
        jade.util.leap.Iterator iter = desc.getAllServices();
        while (iter.hasNext())
        {
            ServiceDescription d = (ServiceDescription)iter.next();

            // generate WSDL from ontology
            generateWSDL(d);

            // register agent in web service store
            registerAgentAsWebService(d,p);
        }
    }
}
```

Výpis 11: Zpracovávání subscriptions z yellow pages.

Způsob generování WSDL je popsán v kapitole 9.10.1. Vraťme se ale zpět k registraci nového agenta. Ve výpisu kódu 11 je popsán způsob, jakým se náš SubscriptionAgent dozví o nově zaregistrovaném agentu. Nyní je potřeba tuto informaci uložit na místo, kam budou mít přístup všechny instance GWHandleru. Uložiště s touto charakteristikou už bylo zmíněno v kapitole 7.1.6, jmenuje se *Application State*. Otázkou ale je, jak se Subscription agent, který je spuštěn v rámci MAS, tedy v podstatě mimo ASP.NET aplikaci, dostane k *Application State*.

Než bude zodpovězena výše uvedená otázku, dovolím si zde popsat metodu, která slouží ke spuštění agenta v MAS. Instance třídy AgentContainer umožňuje zavolat metodu createNewAgent(), která má 3 parametry:

1. název agenta,
2. třída, která reprezentuje agenta,
3. pole objektů, která budou agentovi předány.

A právě poslední parametr je klíčem k odpovědi na otázku „Jak se Subscription agent dostane k objektu v Application State“. Před spuštěním Subscription agenta se vytvoří objekt Hashtable, který je předán Subscription agentovi a zároveň je tento objekt přidán do Application State. Fyzicky(v paměti) se jedná o jeden objekt, nicméně reference na tento objekt jsou předány agentovi a do *Application State*. A sdílené uložiště pojmenované jako *wsStore* je hotovo...

Metoda *registerAgentAsWebService()* tedy přidá do uložiště wsStore informace o novém agentovi. Informace o novém agentovi se tak dostane i mimo MAS - konkrétně je tato informace potřeba v GWHandleru(mimo jiné), který zpracovává příchozí SOAP zprávy.

9.10 Vlastní komponenty

Bylo vytvořeno několik komponent, které jsou nezávislé na celém řešení a mohou tak být použity i na jiných projektech.

9.10.1 Popis rozhraní webových služeb - WSDL

Pro vytváření WSDL definice z ontologie daného agenta je vytvořena samostatná komponenta WSDLGenerator. Tato komponenta má několik metod z nichž nejdůležitější je metoda *getWSDLStream()* s parametry: ontologie, název agenta a URL adresa. Návratovou hodnotou této metody je, jak již název napovídá, Stream reprezentující výslednou WSDL definici. Tento stream je dále zapsán na disk, kde je vytvořené WSDL dostupné skrz URL *http://server:port/nazevAgenta.agent?wsdl*.

Výsledné WSDL odpovídá specifikaci WSDL 1.1 z roku 2001. Nejnovější verze WSDL 2.0 z roku 2007, která byla přijata konsorciem jako doporučení, není zatím příliš využívána.

9.10.2 Seznam dostupných webových služeb - WS-Inspection

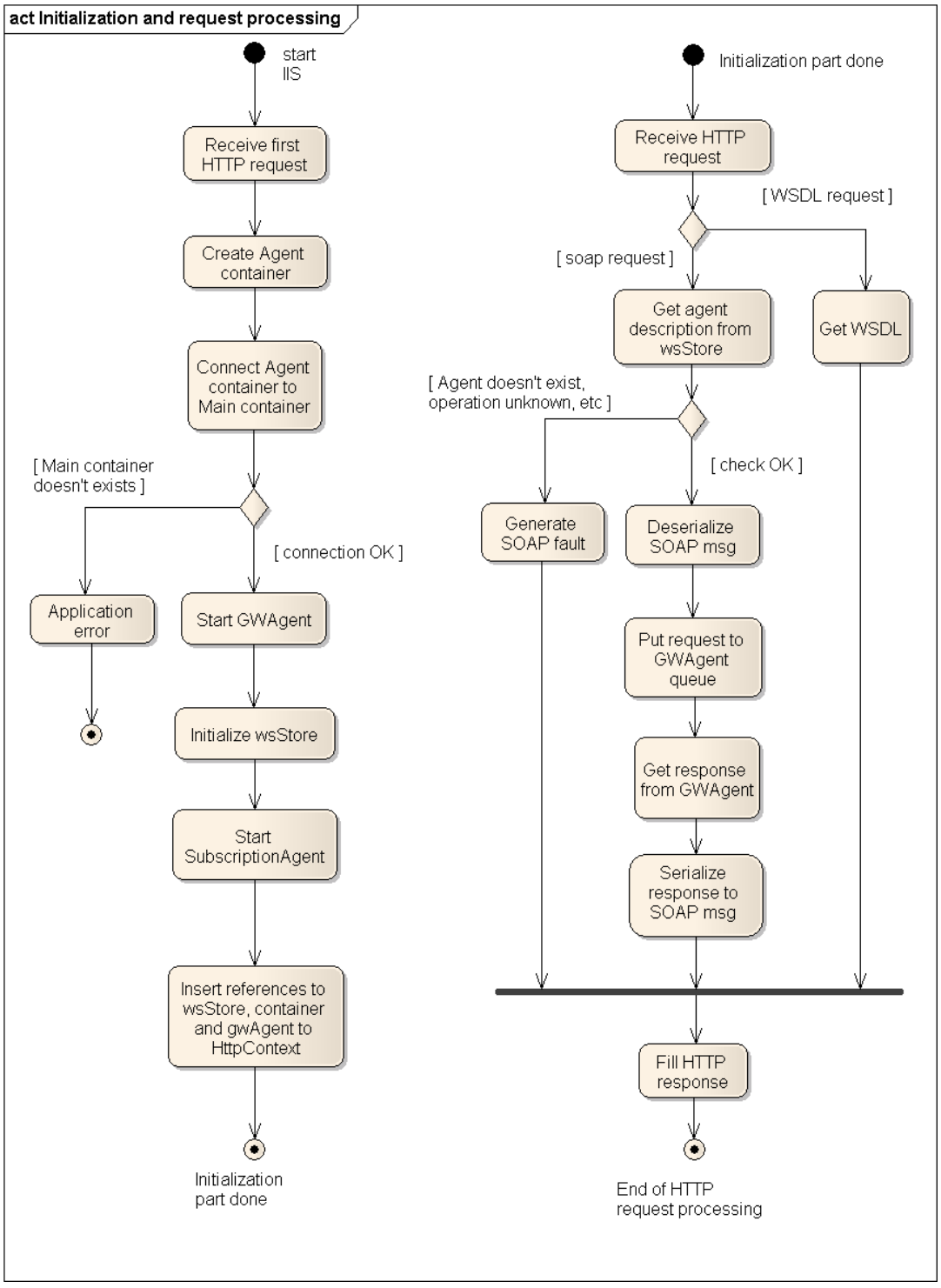
http://server:port/inspection.wsil. Na URL adrese *http://server:port/inspection.wsil* je dostupný seznam všech webových služeb reprezentující konkrétní agenty. Tento seznam je vytvořen

nezávislou komponentou WSILGenerator, která tento seznam vytvoří jako XML soubor dle specifikace WS-Inspection 1.0.

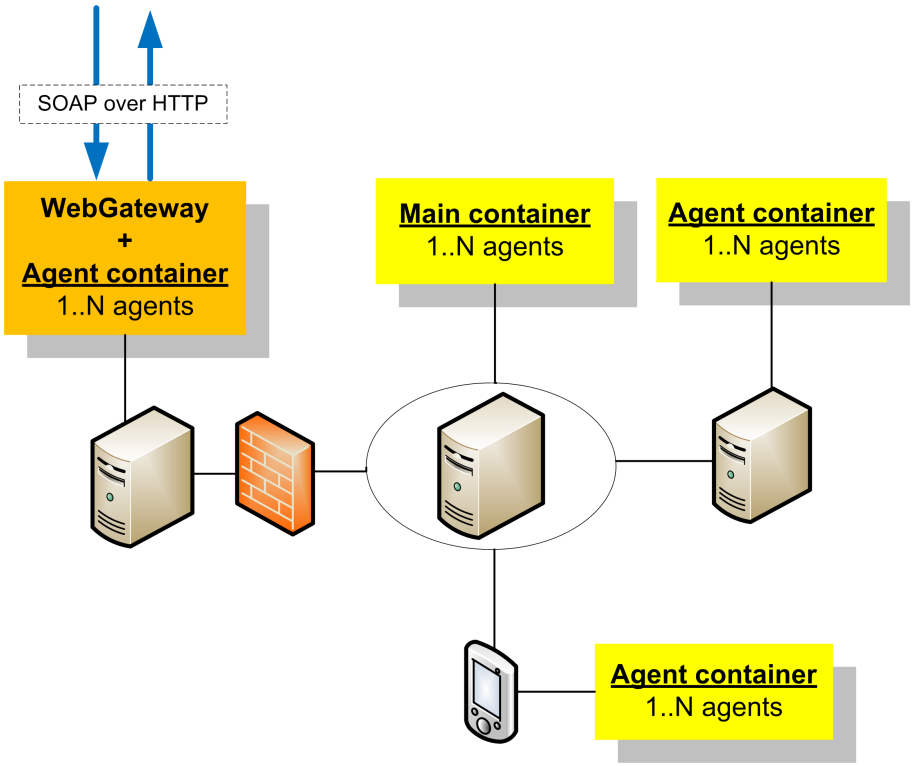
9.11 Logování

Nepsaným pravidlem pro vývoj aplikací je zaznamenávaní událostí do logovacího souboru. Tyto informace poskytují neocenitelnou pomoc ve chvílích, kdy aplikace nefunguje, jak má.

Vyvinutá aplikace používá pro logování technologii Log4Net, která je vyvíjena pod hlavičkou Apache Software Foundation jako sourozenec logovacího nástroje Log4J pro platformu Java.



Obrázek 17: Aktivitní diagram pro proces spuštění aplikace a zpracování požadavku



Obrázek 18: Možnosti reálného použití aplikace s ohledem na architekturu

10 Testování aplikace

Kvalitu výsledného řešení je nutné náležitě otestovat, jelikož se počítá, že aplikace bude využita i v reálném provozu, kde může dojít k většímu zatížení. Proto je vhodné provést kromě funkčního i zátěžové testování.

10.1 Agenti k testování

Celé řešení bylo vyzkoušeno s pomocí dvou vytvořených Agentů. První agent umí sčítat a násobit čísla. Operací druhého agenta je výpočet MD5 otisku textu. Jednotlivé operace byly zvoleny záměrně tak, aby jejich výpočetní složitost neovlivňovala výsledky zátěžového testu celého řešení.

Poznámka 10.1 Operace poskytované agenty použitými pro testování nejsou úplně charakteristické pro Multi-agentní systémy. Nicméně jsou plně dostačující pro otestování funkčnosti celého řešení.

Implementace jednotlivých agentů je díky zvoleným operacím triviální. Nicméně pro správnou funkčnost řešení je nutné provést několik kroků:

Krok č.1 znamená zaregistrovat operace, které budou dostupné prostřednictvím webové služby. Tento krok musí být proveden v metodě *setup()* konkrétního agenta (tzn. při inicializaci agenta) a je zobrazen na výpisu kódu 12.

```
24  ServiceDescription sd = new ServiceDescription();
25  sd.setName(this.getLocalName());
26  sd.addOntologies(ontology.getName());
27
28  // make an agent available through WS
29  sd.addProperties(new Property(GWConst.PUBLIC_AS_WEBSERVICE, true));
30
31  // create list of available operations
32  // operation name string consist: ontologyName.AssemblyName.operationName
33  sd.addProperties(new Property(GWConst.OPERATION, "CalculatorOntology.Agents.Sum"));
34  sd.addProperties(new Property(GWConst.OPERATION, "CalculatorOntology.Agents.Multiply")
        );
```

Výpis 12: Registrace agenta k publikování prostřednictvím WS

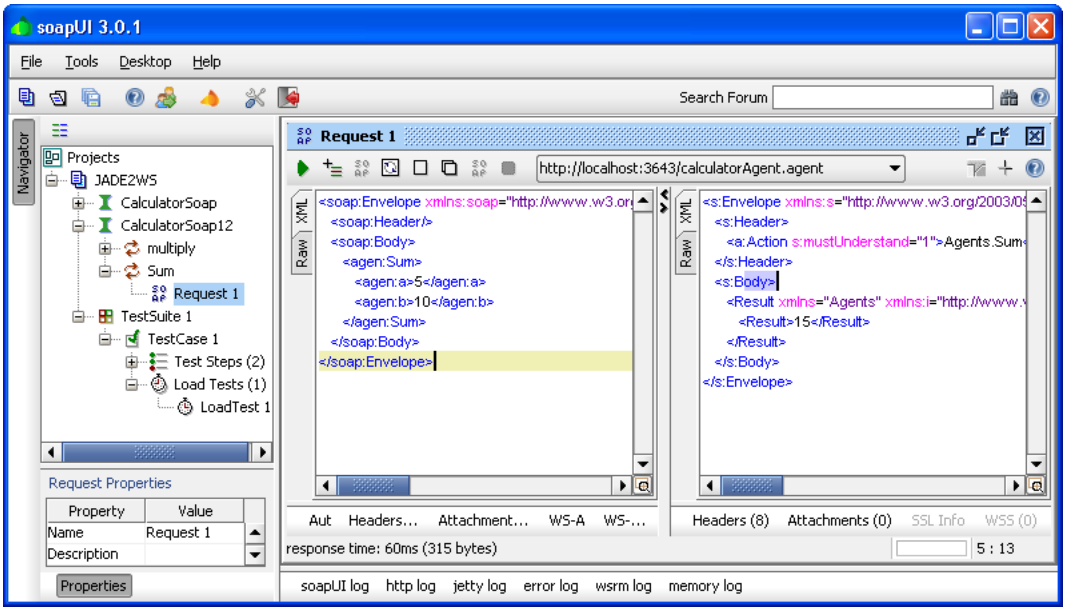
Krok č.2 zahrnuje doplnění anotací *[DataMember]* a *[DataContract]* do tříd reprezentujících ontologii. Třída reprezentující operaci ontologie musí mít anotaci *[DataContract]*. Dále je vhodné k této anotaci přidat parametry jako *Name* (vlastní název operace) a *Namespace*, který bude použit při serializaci a deserializaci SOAP zpráv (více o serializaci/deserializaci v kapitole 9.7). Jednotlivé proměnné třídy reprezentující parametry operace musí být označeny jako „public“ a musí mít anotaci *[DataMember]*.

10.2 Nástroje pro testování

Pro otestování rozhraní webových služeb byl použit open source nástroj **soapUI** v.3.0.1[19].

[19] Free verze tohoto nástroje je ke stažení na adrese `http://www.soapui.org/` (leden 2010)

Tento komplexní nástroj pro testování webových služeb umožňuje simulovat klienta webové služby tak, že uživateli poskytne UI pro úpravu SOAP zpráv. Vytvořené zprávy je samozřejmě možné odeslat na rozhraní webové služby. Samozřejmostí je validace odpovědi - SOAP zprávy, viz obrázek 19.



Obrázek 19: Grafické rozhraní nástroje soapUI

SoapUI rovněž umožňuje z předpřipravených SOAP zpráv vytvořit testovací případy (Test case). Ty mohou být spouštěny jednotlivě anebo se mohou stát součástí celého balíku testovacích případů (Test suite). Vytvořené testovací případy z nástroje soapUI jsou obdobou unit testů, které se používají při testování aplikací na úrovni kódu.

Ani otázka zátěžového testování není pro soapUI tabu. Vytvořené testovací případy mohou být použity také pro zátěžové testování, kdy jsou jednotlivé SOAP zprávy posílány na webovou službu ve více vláknech (dle konfigurace). Průběžné výsledky zátěžového testování jako např. průměrná odezva, maximální odezva, minimální odezva, počet chybných odpovědí, apod. jsou přehledně zobrazeny v tabulce nebo v grafu.

10.3 Použitý hardware a software

Všechny testy proběhly na notebooku Asus M6A s následující hardwarovou konfigurací:

- 1,5 GB RAM (DDR2),
- 5400 ot./min HDD,
- CPU Intel Pentium M 730 (1,6 GHz, 2 MB L2 cache, 533 MHz FSB).

Softwarová konfigurace:

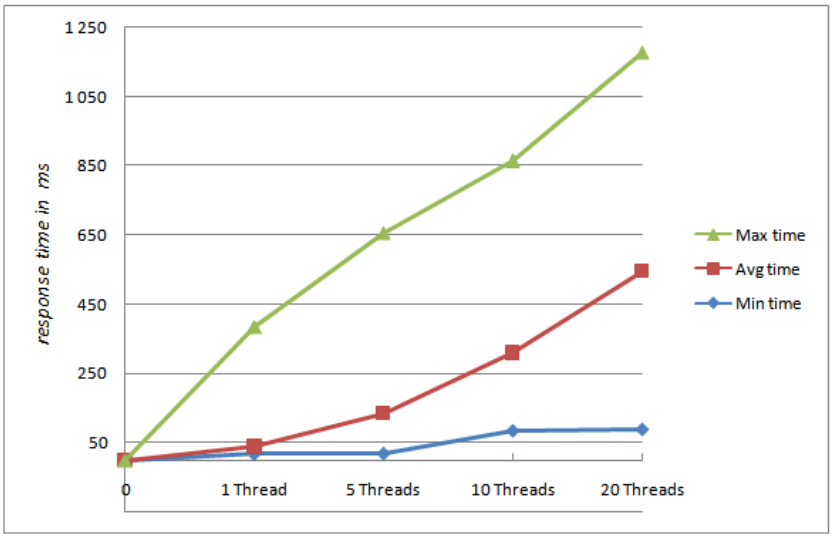
- Operační systém: Windows XP Service Pack 3,
- Aplikační server: ASP.NET Development Server (Visual Studio 2008) + Apache Tomcat 6.

10.4 Zátěžové testování

V nástroji soapUI byly vytvořeny testovací případy pro volání jednotlivých agentů (skrz webové služby). Tyto testovací případy se staly součástí zátěžového testu, jehož cílem je poskytnout informativní hodnoty pro zhodnocení výkonnosti celého řešení.

Byly provedeny celkem 4 testy: 1 test pro sériové volání, další 3 testy pro volání v pěti vláknech, volání v deseti a dvaceti vláknech. Jednotlivé testy trvaly přesně 2 minuty. Tyto 4 testy byly provedeny celkem dvakrát (pro zpřesnění celkového výsledku) a výsledné časy byly zprůměrovány. Výsledek testů je vidět na obrázku 20.

Z pozvolna narůstající křivky Avg.time (průměrný čas) lze soudit, že celé řešení je stabilní i při větším zatížení. Během validace příchozích zpráv nebyla zjištěna žádná chyba, které by naznačovala problémy při paralelním zpracování.

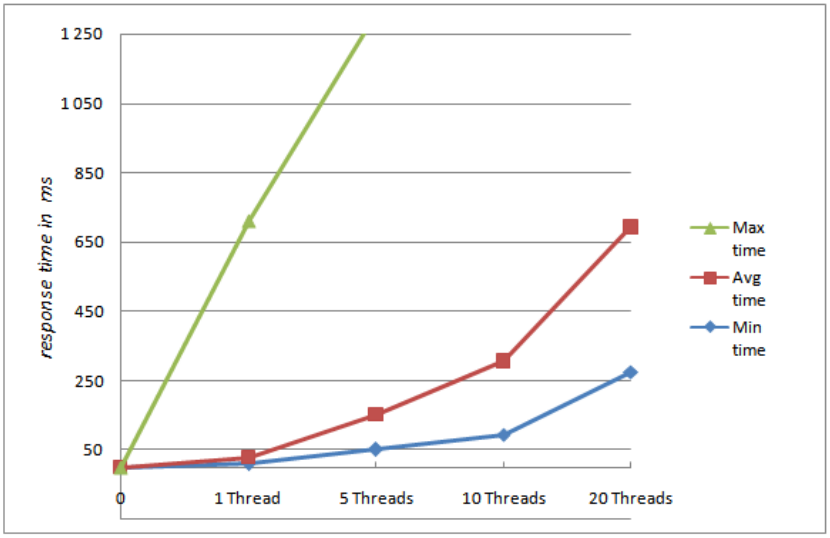


Obrázek 20: Grafické znázornění výsledků výkonnostního testu Jade2WS for .NET

10.5 Porovnání s WSIG

Pro názornost bylo řešení Jade2WS for .NET porovnáno s již existujícím řešením WSIG (napsaný v Javě). Testování proběhlo opět s využitím nástroje soapUI, kdy požadavky byly směrovány na dva agenty. Výsledek testů je zobrazen na obrázku 21.

Z výsledku testu je patrné, že řešení WSIG je mírně pomalejší než mnou vytvořený Jade2WS for .NET. Nicméně rozdíly jsou zanedbatelné (průměrné časy jsou rozdílné o cca 170 ms při zatížení 20-ti vlákny).



Obrázek 21: Grafické znázornění výsledků výkonnostního testu WSIG

10.6 Propojení platforem .NET a Java

Implementované řešení umožňuje zpřístupnit Agenta přes webovou službu nezávisle na tom, zda je naimplementovaný v Javě (Jade-leap) či .NETu (Jade-leap). Jako důkaz tohoto tvrzení byl v Javě naimplementován Agent, jehož jedinou operací je vypočítat MD5 otisk zprávy. Pro tohoto agenta byl vytvořen nový Agent kontejner připojený k hlavnímu kontejneru v .NETu.

Vzhledem k vlastnostem architektury musely být třídy reprezentující ontologii daného agenta vytvořeny dvakrát, jednou v C# a podruhé v Javě. Jedním z důvodů této duplicity je způsob serializace SOAP zpráv, při kterém jsou vyžadovány třídy dané ontologie napsané v C#, viz kapitola 9.7.

Funkčnost propojení byla opět otestována jednoduchým testem s pomocí nástroje soapUI. Jednotlivé odezvy byly v řádech desítek milisekund a během dvou minutového testu nebyl zaznamenán žádný problém.

10.7 Komunikace s webovými službami - WSInvokerAgent

Pro ověření funkčnosti WSInvoker Agenta byl naimplementován následující scénář:

- Agent, nazvaný jako WeatherAgent, potřebuje pro svou práci informace o aktuálním počasí,
- Informace o aktuálním počasí poskytuje webová služba na URL `http://www.deeptraining.com/webservices/weather.asmx?WSDL`,
- WeatherAgent komunikuje s webovou službou prostřednictvím WSInvoker Agenta.

WeatherAgent kontroluje počasí co 2 minuty. Výsledek, tzn.předpověď počasí, zapíše do souboru /logs/ws_log.txt.

Poznámka 10.2 Uvedená webová služba byla vybrána zcela náhodně. Dostupnost této webové služby není nikterak zaručena. Nicméně pro otestování funkcionality svůj účel splnila. Funkčnost komunikace s touto webovou službou skrz Weather Agenta demonstruje video na přiloženém CD ve složce /documentation.

11 Možnosti nasazení v praxi

Výsledkem této práce je řešení pro komunikaci multi-agentních systémů a webových služeb, které využívá nejnovější technologie z frameworku .NET v.3.5 jako např. Windows Communication Foundation (WCF), apod. Cílem bylo vytvořit řešení, které bude použitelné a také jednoduše rozšiřitelné. Nechť čtenář (případně vývojář) posoudí, zda se výsledek shoduje s původním záměrem.

Celé řešení brány mezi MAS a WS je nasaditelné v praxi a to i přesto, že otázka bezpečnosti není v této práci nijak zohledněna. Zabezpečení přístupu k webovým službám (reprezentující jednotlivé agenty) lze efektivně vyřešit např. pomocí autentizace klientskými certifikáty na úrovni transportní vrstvy (https). Toto zabezpečení lze zajistit na úrovni IIS serveru, tzn. není nutno upravovat zdrojové kódy aplikace.

Provedené testy ukázaly, že celé řešení je jednoduše připojitelné k MAS skládajícího se z Agent kontejnerů implementovaných na různých platformách (Java nebo .NET). Jediným omezením tedy je nutnost spustit Jade2WS for .NET na IIS serveru, který nemusí být vždy dostupný (např. z důvodu použití některé z linuxových distribucí namísto Microsoft Windows).

12 Problémy a strasti při vývoji aplikace

Při vývoji této aplikace, která je postavena z technologií dvou odvěkých rivalů - .Net frameworku a Javy, jsem musel čelit mnohým nástrahám, které vývoj aplikace výrazně ztížily.

12.1 Nedostupnost některých metod

Jedním z příkladů je nedostupnost metody Ontology.getActionNames() v Jade-LEAP (informace poskytnuté touto metodou jsou nutné např. pro generování WSDL, metoda je v Jade přístupná).

Řešení: Tento nedostatek musel být řešen zásahem do zdrojových kódů frameworku Jade a následnou rekompilací Jade frameworku.

Možné následky: V případě přechodu na novější verzi frameworku Jade je nutné tento zásah opakovat.

12.2 Nedostupnost některých konstant

Z dalších nástrah lze uvést např.nedostupnost konstant jako např. IntrospectionVocabulary.BORNAGENT, BasicOntology.INTEGER.

Řešení: vyřešeno použitím konkrétních řetězců, které tyto konstanty zastupují, tzn. namísto BasicOntology.INTEGER je v aplikaci použito textového řetězce „int“.

Možné následky: Nekompatibilita s novějšími verzemi frameworku Jade a tudíž i nefunkčnost celé brány.

12.3 Ukončení vývoje a podpory Visual J#

Riziko možných problémů do budoucna rapidně zvyšuje prohlášení Microsoftu, který v lednu 2007 prohlásil, že Microsoft Visual J# (tj. nástroj nutný k portaci Jade-LEAP do prostředí .NET), již nebude dále rozvíjet a **podpora Visual J# 2.0 skončí v druhé čtvrtině roku 2015**[20].

Řešení: V tuto chvíli neznámé. Jednou z možných alternativ pro případ budoucích problémů, je použití brány vytvořené v javě.

Možné následky: V momentě, kdy tvůrci frameworku Jade začnou používat některé novější prvky Javy 6 nebo dokonce Javy 7 (jako např. anotace), je Jade-LEAP pro .NET a taky celá tato aplikace v podstatě odsouzena k pomalému zániku. Na jaře 2010 by měla být vydána nová verze Jade 4.0 (aktuální verze je 3.7) a je tedy otázkou, zda bude ještě možné tuto novou verzi převést do prostředí .NET.

[20] Prohlášení je dostupné na URL `http://msdn.microsoft.com/en-us/vjsharp/default.aspx` (leden 2010)

12.4 Shrnutí

Potencionální uživatel-vývojář by měl ve světle výše uvedených fakt zvážit, zda použije řešení brány mezi MAS a WS, a potažmo celý framework Jade-LEAP (.NET verzi) pro projekty s dlouhodobou působností na trhu.

13 Další rozvoj aplikace

Každá softwarová aplikace má svůj životní cyklus, který začíná sběrem požadavků na aplikaci a pokračuje stejně jako kapitoly tohoto dokumentu, tj. návrhem řešení, implementací, testováním atd. Dle vodopádového modelu vývoje aplikací se nyní nachází toto řešení na úrovni procesu údržby a podpory (maintenance & support). Tento často opomíjený a opovrhovaný proces má obrovský vliv na vnímání celého řešení vývojáři či dokonce koncovými uživateli. Je naivní doufat v bezchybnost a dokonalost celé aplikace, i když úspěšně prošla všemi testy. Každá aplikace ukáže své kvality až při reálném využití, do té doby se dá s nadsázkou hovořit o softwarovém prototypu. Dalším z důvodů, proč by neměl být proces údržby a podpory opomíjen, jsou možné změny na trhu, jako např. nové verze použitých knihoven (v tomto případě frameworku Jade), jiné nároky na zabezpečení, apod. Jsem si je vědom, že vyvinout použitelnou aplikaci v rámci diplomové práce znamená jistý druh zodpovědnosti do budoucna - nelze vyvinout aplikaci a poté ji nechat skomírat do doby než přijde někdo jiný s lepším řešením. To je také jedním z důvodů, proč bude celé řešení zveřejněno jako doplněk (add-on) pro framework Jade. Zdrojové kódy budou volně dostupné (open-source) pod licencí LGPL. Je-li toto řešení opravdu užitečné a kvalitní, jistě si najde svou komunitu vývojářů, která ho bude dále vyvíjet (třeba i pod mým vedením). Jedině tak lze zajistit budoucnost celé aplikace.

14 Závěr

Cílem této práce bylo nastudovat problematiku MAS a jejich implementace frameworkem Jade natolik, aby bylo možné vyvinout řešení poskytující propojení mezi webovými službami a MAS. Přes počáteční obavy plynoucí z neznalosti MAS, frameworku Jade a dokonce i .NET frameworku se povedlo vyvinout provozuschopné řešení, které může být použito na jiných projektech.

Během psaní této diplomové práci jsem si všechny své poznatky, nápady a náměty k dalšímu vývoji zaznamenával do tzv. myšlenkové mapy, která může sloužit jako tzv. knowledge base pro další zájemce o studium frameworku Jade. Myšlenková mapa je vytvořena v programu Free Mind Map (soubor je umístěn na přiloženém CD).

Nejen pro potřeby vedoucího diplomové práce a oponenta je na přiloženém CD dostupné několik videí ve formátu SWF (Adobe Flash), kde demonstruji jak instalování aplikace, tak samotné ověření funkčnosti.

Na samém počátku této práce stála má zvědavost, co se skrývá pod spojením tří slov „Multi-agentní systémy“. Tato zvědavost mi poskytla rozumný náhled do světa technologií, které mají nadějnou budoucnost. Spolu v kombinaci s přívětivosti frameworku Jade ve mě tato práce vzbudila zájem o celý koncept MAS, který bude dále rozvíjen i po úspěšném obhájení této práce. Jedním z kroků, které chci podniknout je zveřejnění této práce včetně zdrojových kódů a následná spolupráce s komunitou frameworku Jade na jejím dalším rozvoji.

15 Literatura

[1] KUBÍK, Aleš. *Agentově-orientované inženýrství : nové paradigma pro tvorbu softwaru?*. [s.l.], 2003. 16 s. Referát. Ústav informatiky, Slezská univerzita, Opava. Dostupné z WWW: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.83.7237&rep=rep1&type=pdf (březen 2010).

[2] SISLAK, David, et al. *AGENTFLY: a Multi-Agent Airspace Test-bed*. [s.l.] : [s.n.], 2008. 2 s. Dostupný z WWW: http://agents.felk.cvut.cz/cgi-bin/docarc/public.pl/document/192/AAMAS08_demo5.pdf (leden 2010). ISBN 9780981738130.

[3] The Foundation for Intelligent Physical Agents (FIPA). *FIPA specifications* [online]. 2005. FIPA, c2005 [cit. 2010-01-14]. Specifikace FIPA. Angličtina. Dostupný z WWW: http://www.fipa.org/specifications/index.html.

[4] BOOTH, David, et al. *Web Services Architecture : W3C Working Group Note 11 February 2004* [online]. 2004 [cit. 2010-01-30]. Dostupný z WWW: http://www.w3.org/TR/ws-arch/.

[5] CHRISTENSEN, Erik, et al. *Web Services Description Language (WSDL) 1.1*. [s.l.] : W3C, 2001. 33 s. Dostupný z WWW: http://www.w3.org/TR/wsdl.

[6] BALLINGER, Keith, et al. *Web Services Inspection Language (WS-Inspection) 1.0*. [s.l.] : International Business Machines Corporation, Microsoft, c2001. 19 s. Dostupný z WWW: http://download.boulder.ibm.com/ibmdl/pub/software/dw/specs/ws-wsilspec/ws-wsilspec.pdf.

[7] MENŠÍK, Marek, ČÍHALOVÁ, Martina. *KQML vs ACL*. LabIS [online]. neznámý, č. neznámé [cit. 2010-02-02]. Dostupný z WWW: http://labis.vsb.cz/labis/files/dokumentace/m-tina/kqml_vs_acl.doc.

[8] BELLIFEMINE, Fabio, CAIRE, Giovanni, GREENWOOD, Dominic. *Developing Multi-Agent Systems with JADE*. West Sussex : John Wiley, 2007. 303 s. ISBN 0470057475.

[9] KESSLER, Robert, RUSITSCHKA, Steffen. *Building Agents with Visual Studio .NET*. AAMAS⁄03 [online]. 2003 [cit. 2010-01-14]. Dostupný z WWW: http://www.cs.utah.edu/arg/papers/agentvsnet.doc.

[10] BELLIFEMINE, Fabio, et al. *Jade Administratorś Guide*. JADE Board : [s.n.], 2007. 37 s. Dostupný z WWW: http://jade.tilab.com/doc/administratorsguide.pdf(říjen 2008).

[11] Sun Microsystems.*Java Servlet API Specification - Version 2.1* [online]. 2007 [cit. 2009-11-01]. Dostupný z WWW: http://java.sun.com/products/servlet/2.1/api/javax.servlet.Servlet.html.

[12] Microsoft. *ASP.NET Application Life Cycle Overview* [online]. 2009 [cit. 2009-11-20]. Angličtina. Dostupný z WWW: http://msdn.microsoft.com/en-us/library/ms178473(VS.80).aspx.

[13] Microsoft. *ASP.NET Application State Overview* [online]. c2009 [cit. 2009-11-01]. Dostupný z WWW: http://msdn.microsoft.com/en-us/library/ms178594.aspx.

[14] Microsoft. *ASP.NET Global.asax Overview* [online]. c2009- [cit. 2009-11-05]. Dostupný z WWW: http://msdn.microsoft.com/en-us/library/1xaas8a2(VS.71).aspx.

[15] VONDRÁK, Ivo, KOŽUSZNIK, Jan, OCHODKOVÁ, Eliška. *Metody specifikace softwarových systémů : pro kombinované a distanční studium*. Ostrava : [s.n.], 2006. 77 s. Dostupný z WWW: http://www.cs.vsb.cz/kozusznik/vyuka/mss/MSS2006.pdf.

[16] MUSIL, Marek. Multi-agentní platforma pro podporu vyhledávání. [s.l.], 2008. 40 s. Masarykova univerzita, Fakulta Informatiky. Vedoucí bakalářské práce Mgr.Matěj Štefaník. Dostupný z WWW: http://is.muni.cz/th/173455/fi_b/Bc.pdf.

[17] BOX, Don, et al. *Simple Object Access Protocol (SOAP) 1.1 : W3C Note 08 May 2000* [online]. NOTE-SOAP-20000508 . W3C, 2000 [cit. 2009-01-12]. Angličtina.Dostupný z WWW: http://www.w3.org/TR/2000/NOTE-SOAP-20000508/.

[18] Microsoft. *XML Serialization in the .NET Framework* [online]. 2003 [cit. 2009-11-15]. Angličtina. Dostupný z WWW: http://msdn.microsoft.com/en-us/library/ms950721.aspx.

[19] LOWY, Juval. *Programming WCF Services.* [s.l.] : OŔeilly, 2007. 634 s. ISBN 0-596-52699-7.

A Zprovoznění aplikace

Předpoklady:

- Nainstalovaný IIS 7.5 (Windows 7, Windows Server 2008) včetně IIS 6 Metabase Compatibility a Application development features - ASP.NET,
- Nainstalovaný .NET framework 3.0 a výšší,
  - aplikace má svůj vlastní web.config s definovanými HTTPHandlery, proto je nutné povolit tuto vlastnost v IIS,
- Nainstalovaný Visual J# 2.0.

Pro instalace aplikace Jade2WS for .NET stačí spustit instalátor setup.exe, který je dostupný ve složce /install na přiloženém CD.

Pro názornost je postup instalace vysvětlen ve formě videa ve formátu SWF (Adobe Flash). Toto video lze nalézt ve složce /documentation/how_to_install.swf.

Poznámka A.1 Při vytváření výukových videí jsem se často potýkal s chybou „StaleProxyException Class SubscriptionAgent not found“.
Tato chyba se objevovala jen na testovacím prostředí Windows 7 a IIS 7.5. Řešením této chyby je jakýkoliv zásah do web.config instalované aplikace (např. přehození pořadí assembly elementů). Přesná příčina tohoto problému nebyla nalezena.

B Zdrojové kódy

Všechny zdrojové kódy řešení „Jade2WS for .NET“ jsou v adresář src na přiloženém CD. Tento adresář obsahuje celé řešení (solution), které lze snadno importovat do vývojového prostředí Microsoft Visual Studio 2008.

Solution je rozděleno na následující projekty:

- **Agents** - testovací agenti jako Weather Agent, HashMaster Agent, Calculator Agent,
- **Core** - Hlavní kontejner, který musí být spuštěn jako první,
- **DynamicWSInvoker** - Nezávislá komponenta umožňující dynamické volání webových služeb,
- **GatewayUtil** - obsahuje nutné části celého řešení, jako např. GatewayAgent, Subscription Agent, WSInvoker Agent, Gateway Constants, apod.,
- **Ontology2WSDL** - nezávislá komponenta umožňující vytvoření WSDL definice z ontologie,
- **Web** - ASP.NET aplikace, bez které by nebylo možné zavolat konkrétního agenta skrz webovou službu, atd.,
- **WSInspection** - nezávislá komponenta sloužící pro vygenerování WSIL souboru (tzv.registr webových služeb).

C Kompilace frameworku Jade pro platformu .NET

Jelikož je framework Jade dostupný pouze pro platformu Java, je nutné jej převést do podoby, která bude využitelná i na platformě .NET (dll knihovna). Předem upozorňuji, že kompilace zdrojových kódů Jade do .NET frameworku je poměrně obtížná. Navíc se může stát, že kompilace některé z novějších verzí frameworku Jade se již nemusí podařit. Nástroj nutný ke kompilaci (Microsoft Visual J# 2.0) již nebude dále rozvíjen.

Před prvotní kompilaci je nutné se obrnit trpělivostí a vyhradit si cca 1-2 hodiny svého času.

C.1 Nutný software

- .NET framework 1.1,
- Microsoft Visual J# 2.0 Redistributable Package - SE,
- Ant,
- JDK 1.5 a vyšší.

C.2 Adresářová struktura

Je nutné stáhnout LEAP-addon a Jade, poté vytvořit následující adresářovou strukturu

```
1 Jade
2     |– src  (adresář se  zdrojovými kódy Jade)
3     |– leap
```

Poté je nutné upravit buildLEAP.propertis ve složce leap a nastavit cestu k .NET frameworku. a pak jen spustit příkaz *ant dotnet rebuild* Pokud vše proběhne bez chyby (což se při prvním spuštění nedá moc očekávat), bude výsledné DLL dostupné v adresáři Jade/leap/dotnet/lib.

C.3 Problémy s kompilací jednotlivých verzí

Každá verze Jade přináší nové problémy při kompilaci z Javy do .NETu.

C.3.1 Jade 3.5

viz příspěvek
`http://sharon.cselt.it/pipermail/jade-develop/2008q2/012396.html`

C.3.2 Jade 3.7

Postup pro kompilaci vyžaduje zásah do zdrojových kódů viz
`http://sharon.cselt.it/pipermail/jade-develop/2009q4/014525.html`
`http://sharon.cselt.it/pipermail/jade-develop/2009q4/014526.html`

C.3.3 Jak přidat javovské třídy do kompilace

Standardně jsou jenom některé třídy z frameworku Jade zkompilovány do DLL knihovny. Občas může být potřeba přidat do této DLL knihovny i jiné třídy. Abychom tohoto docílili je nutné editovat soubor dotnet.xml a přidat do elementu *fileset* následující elementy s cestou k javovským třídám, viz následující ukázka

```
1  <fileset dir="${jade-src}">
2          <include name="**/jade/wrapper/gateway/JadeGateway.java"/>
3          <include name="**/jade/wrapper/gateway/GatewayAgent.java"/>
4          <include name="**/jade/wrapper/gateway/GatewayBehaviour.java"/>
```

C.4 Další návody pro kompilaci

Kompilace je také popsána na straně 18 publikace LEAP User Guide
`http://jade.tilab.com/doc/tutorials/LEAPUserGuide.pdf` (leden 2010)

Další návod lze nalézt například na URL
`http://casal.upc.es/~marti23/JadeOverNET.shtml` (leden 2010)